Universität Stuttgart

Stable and Mass-Conserving High-Dimensional Simulations with the Sparse Grid Combination Technique for Full HPC Systems and Beyond

Vom Stuttgarter Zentrum für Simulationswissenschaften (SC SimTech) und der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Theresa Pollinger

aus Eichstätt

Hauptberichter*in:	Prof. Dr. Dirk Pflüger
Mitberichter*in:	Prof. Dr. Philipp Neumann
Tag der mündlichen Prüfung:	26. Januar 2024

Institut für Parallele und Verteilte Systeme

2024

Contents

Zusammenfassung

Angesichts der fortschreitenden Klimakrise hat die kontrollierte Plasmafusion das Potenzial, eine der entscheidenden wissenschaftlichen Errungenschaften des 21. Jahrhunderts zu werden. Um die turbulenten Felder in magnetisch einschließenden Fusionsreaktoren zu verstehen, war und ist Computersimulation sowohl eine Bereicherung als auch eine Herausforderung. Der einschränkendste Faktor für groß angelegte, hochgenaue prädiktive Simulationen liegt im Fluch der Dimensionalität, der alle gitterbasierten Beschreibungen von Plasma auf Grundlage der Vlasov–Poisson- und Vlasov–Maxwell-Gleichungen prägt. In der vollständigen Formulierung ergeben sich sechsdimensionale Gitter und feine Skalen, die aufgelöst werden müssen, was zu nicht-bewältigbaren Anzahlen an Freiheitsgraden führen kann. Typische Ansätze für dieses Problem – Koordinatentransformationen wie Gyrokinetik, Gitteranpassung, Beschränkung auf begrenzte Auflösungen – gehen den Fluch der Dimensionalität nicht direkt an, sondern *um*gehen ihn.

Die *Dünngitter-Kombinationstechnik*, die den Kern dieser Arbeit bildet, ist ein Multiskalen-Ansatz, der den Fluch der Dimensionalität für zeitschrittbasierte Simulationen lindert: Es werden mehrere reguläre gitterbasierte Simulationen ausgeführt, die im Laufe der Simulation gegenseitig Informationen austauschen. Die vorliegende Arbeit verbessert den bisherigen Stand der Kombinationstechnik auf drei Arten: Die Einführung von Massenerhaltung und numerischer Stabilität durch die Verwendung besser geeigneter Multiskalen-Basisfunktionen, Optimierung des Codes für große HPC-Systeme und Erweiterung der Kombinationstechnik auf den weitverteilten Fall mit mehreren HPC-Systemen.

Diese Arbeit analysiert zunächst die häufig verwendete hierarchische Hutfunktion aus der Sicht biorthogonaler Wavelets, was es ermöglicht, die hierarchische Hutfunktion auf einfache Weise durch andere Multiskalenfunktionen (wie die massenerhaltenden CDF-Wavelets) zu ersetzen. Die in der Dissertation vorgestellten

numerischen Studien zeigen, dass dies nicht nur zur Erhaltung führt, sondern auch die Genauigkeit erhöht und numerische Instabilitäten vermeidet—die zuvor ein großes Hindernis für groß angelegte Vlasov-Simulationen mit der Kombinationstechnik darstellten.

Zweitens wurde das Open-Source-Framework DisCoTec erweitert, um die Kombinationstechnik auf den verfügbaren Speicher ganzer Supercomputing-Systeme zu skalieren. DisCoTec ist so konzipiert, dass die Kombinationstechnik um bereits bestehende gitterbasierte Löser gelegt werden kann und nutzt die inhärente Parallelität der Kombinationstechnik. Neben mehreren anderen Beiträgen wurden im Rahmen dieser Arbeit verschiedene kommunikationsvermeidende Multiskalen-Reduktionsschemata entwickelt und in DisCoTec implementiert. Die Skalierbarkeit des Ansatzes wird durch eine umfangreiche Reihe von Messungen in dieser Arbeit bestätigt: DisCoTec lässt sich nachweislich auf die volle Systemgröße von vier deutschen Supercomputern skalieren, einschließlich der drei CPU-basierten Tier-0/Tier-1-Systeme.

Drittens wurde die Kombinationstechnik auf den weitverteilten Fall erweitert, bei dem zwei HPC-Systeme synchron eine gemeinsame Simulation durchführen. Dies wird durch Übertragen von Dateien sowie ausgefeilte Algorithmen zur Aufteilung der verschiedenen Simulationsinstanzen auf die Systeme ermöglicht. Zwei solche Algorithmen wurden im Rahmen dieser Arbeit entwickelt. Durch die drastische Reduzierung des Kommunikationsvolumens, die sich daraus ergibt, wurden erstmals tolerierbare Übertragungszeiten für Kombinationstechnik-Simulationen auf mehreren verschiedenen HPC-Systemen gleichzeitig erreicht.

Diese drei Fortschritte—verbesserte numerische Eigenschaften, effiziente Skalierung auf volle Systemgrößen und die Möglichkeit, die Simulation über ein einzelnes System hinaus auszudehnen—zeigen, dass die Dünngitter-Kombinationstechnik einen vielversprechender Ansatz aufzeigt, um zukünftige hochgenaue Simulationen von höherdimensionalen Problemen, wie Plasmaturbulenz, durchzuführen.

Abstract

In the light of the ongoing climate crisis, mastering controlled plasma fusion has the potential to be one of the pivotal scientific achievements of the 21st century. To understand the turbulent fields in confined fusion devices, simulation has been and continues to be both an asset and a challenge. The main limiting factor to large-scale high-fidelity predictive simulations lies in the Curse of Dimensionality, which dominates all grid-based discretizations of plasmas based on the Vlasov–Poisson and Vlasov–Maxwell equations. In the full formulation, they result in six-dimensional grids and fine scales that need to be resolved, leading to a potentially untractable number of degrees of freedom. Typical approaches to this problem—coordinate transformations such as gyrokinetics, grid adaptation, restricting oneself to limited resolutions—do not directly address the Curse of Dimensionality, but rather work *around* it.

The *sparse grid combination technique*, which forms the center of this work, is a multiscale approach that alleviates the curse of dimensionality for time-stepping simulations: Multiple regular grid-based simulations are run and update each other's information throughout the course of simulation time. The present thesis improves upon the former state-of-the-art of the combination technique in three ways: introducing conservation of mass and numerical stability through the use of better-suited multiscale basis functions, optimizing the code for large-scale HPC systems, and extending the combination technique to the widely-distributed setting.

Firstly, this thesis analyzes the often-used hierarchical hat function from the viewpoint of biorthogonal wavelets, which allows to replace the hierarchical hat function by other multiscale functions (such as the mass-conserving CDF wavelets) in a straightforward manner. Numerical studies presented in the thesis show that

this not only introduces conservation but also increases accuracy and avoids numerical instabilities—which previously were a major roadblock for large-scale Vlasov simulations with the combination technique.

Secondly, the open-source framework DisCoTec was extended to scale the combination technique up to the available memory of entire supercomputing systems. DisCoTec is designed to wrap the combination technique around existing grid-based solvers and draws on the inherent parallelism of the combination technique. Among several other contributions, different communication-avoiding multiscale reduction schemes were developed and implemented into DisCoTec as part of this work. The scalability of the approach is asserted by an extensive set of measurements in this thesis: DisCoTec is shown to scale up to the full system size of four German supercomputers, including the three CPU-based Tier-0/Tier-1 systems.

Thirdly, the combination technique was further extended to the widely-distributed setting, where two HPC systems synchronously run a joint simulation. This is enabled by file transfer as well as sophisticated algorithms for assigning the different simulation instances to the systems, two of which were developed as part of this work. By the resulting drastic reductions in the communication volume, tolerable transfer times for combination technique simulations on different HPC systems have been achieved for the first time.

These three advances—improved numerical properties, scaling efficiently up to full system sizes, and the possibility to extend the simulation beyond a single system—show the sparse grid combination technique to be a promising approach for future high-fidelity simulations of higher-dimensional problems, such as plasma turbulence.

Chapter 1

Motivation and Introduction: Importance and Challenges of High-Fidelity Plasma Simulations

Generating low-carbon energy from plasma fusion has been one of the big visions of the 20th century. Virtually unlimited energy by controlled plasma fusion is a promise that was first formulated in the 1950s, culminating in statements that may seem overly optimistic from today's perspective [139]:

> 'It is the firm belief of many of the physicists actively engaged in controlled fusion research in this country that all of the scientific and technological problems of controlled fusion will be mastered–perhaps in the next few years.'

Although the 1970s and 1980s did see a surge in controlled fusion research funding, most notably in the U.S. [109], the overall public investment was dwarved by research funding in the areas of nuclear fission and fossil energy sources [23]. There is an old joke that states that net positive controlled fusion energy is always n years ahead, where n typically ranges from 10 to 50 (examples in [5, 37, 148]). Still, while we have not reached sustained net-positive fusion energy in 2023, progress with the reactors JET [43] and ITER [24] makes it seem realistic that this goal will eventually be achieved.

Predictive simulation of plasma microturbulence was and continues to be one of the assets—and challenges—for these experiments. After all, one generally doesn't want to build an entire reactor just to see it destroyed due to a plasma disruption, but rather wants the reactor to work as efficiently as possible [150, 161]. This is enabled by high-fidelity simulations of the plasma turbulence in confined fusion reactors.

1.1. Solving the Vlasov–Poisson and Vlasov–Maxwell Systems of Partial Differential Equations

In modeling plasma turbulence, the Vlasov equation [164, 165] is a partial differential equation (PDE) that describes the kinetic motion of the plasma through the phase space of position and velocity, equivalently to the collisionless Boltzmann equation [78][1].

There are several approaches to simulating plasma turbulence, which all solve the Vlasov equation with varying model assumptions and corresponding simplifications. The most ‘coarsely-resolved’ approach is given by the magnetohydrodynamic model, which describes the plasma as a fluid in a state of quasi-equilibrium by a few macroscopic quantities [167]. However, these model assumptions are not applicable to the high field intensities of hot fusion plasmas in the center of a reactor. Another approach are the particle-in-cell (PIC) or Lagrangian particle methods, where plasma particles are treated like in an n-body simulation, and the macro-quantities of the simulation are represented on a grid. While PIC simulations live in $3D$ position space, they inherit the challenges from both particle- and continuum-based methods, such as their inherent numerical noise [117]. (Sparse grid methodology has been shown to appropriately deal with some of these challenges as well [115, 145].)

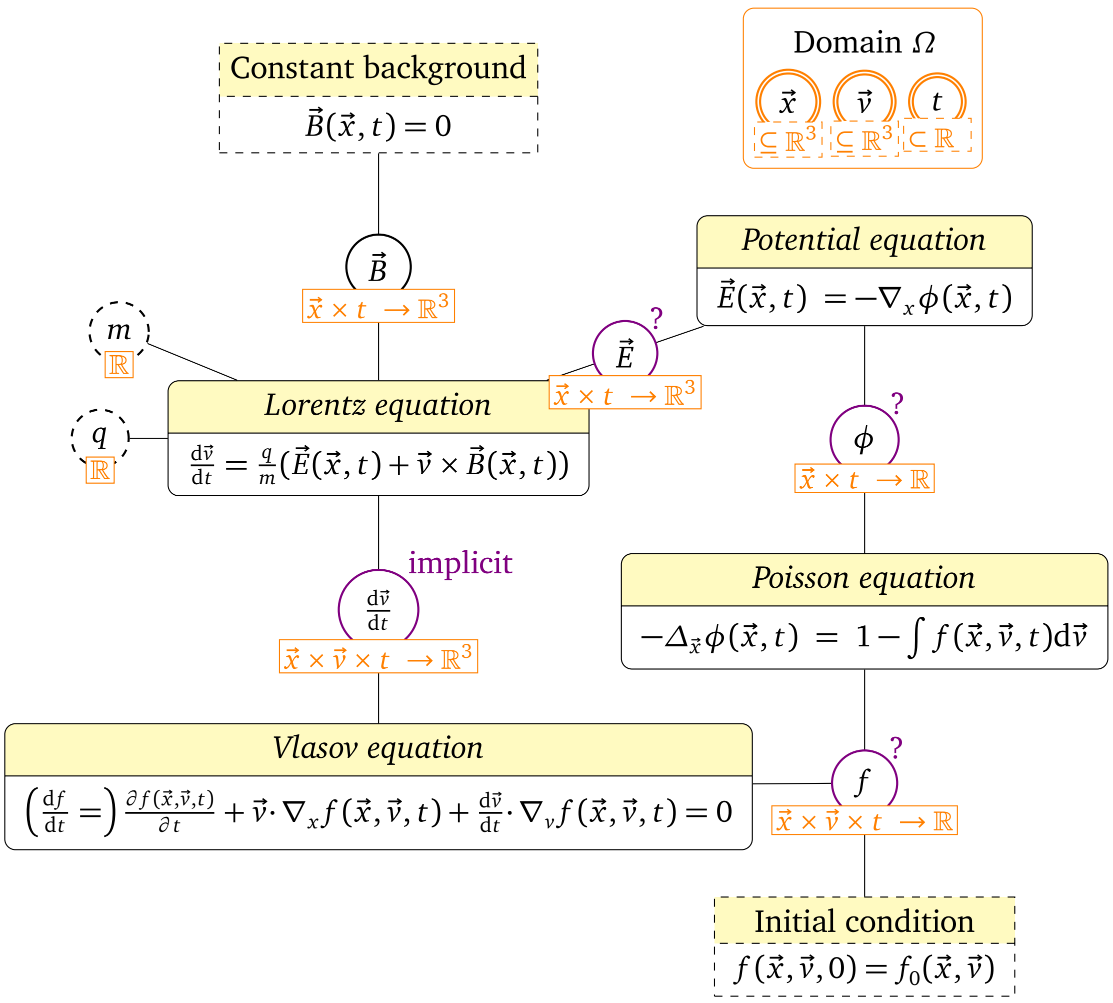
This thesis focuses on the most finely-resolved approach: high-fidelity grid-based direct Vlasov simulations, which are ‘haunted’ by the Curse of Dimensionality.

In its most general form, the Vlasov (collisionless) equation reads

$$\frac{\partial f}{\partial t} + \nabla_{\vec{x}} f \cdot \frac{\mathrm{d}\vec{x}}{\mathrm{d}t} + \nabla_{\vec{v}} f \cdot \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = 0. \tag{1.1}$$

The unknown particle density f denotes the probability of finding a particle at position $\vec{x}$ with velocity $\vec{v}$ at time t. This particle density changes depending on the magnetic and electric fields, and can also influence them.

[1]Henon [78] also lists a variety of other names under which the equation is and had already been known when Vlasov first published his work in 1967.



Domain Ω
$\vec{x}$
$\vec{v}$
t
$\subseteq \mathbb{R}^3$
$\subseteq \mathbb{R}^3$
$\subset \mathbb{R}$
Constant background
$\vec{B}(\vec{x},t)=0$
$\vec{B}$
$\vec{x}\times t \to \mathbb{R}^3$
Potential equation
$\vec{E}(\vec{x},t) = -\nabla_x\phi(\vec{x},t)$
m
$\mathbb{R}$
?
$\vec{E}$
$\vec{x}\times t \to \mathbb{R}^3$
Lorentz equation
q
$\mathbb{R}$
$\frac{d\vec{v}}{dt} = \frac{q}{m}(\vec{E}(\vec{x},t)+\vec{v}\times\vec{B}(\vec{x},t))$
?
ϕ
$\vec{x}\times t \to \mathbb{R}$
implicit
$\frac{d\vec{v}}{dt}$
$\vec{x}\times\vec{v}\times t \to \mathbb{R}^3$
Poisson equation
$-\Delta_{\vec{x}}\phi(\vec{x},t) = 1-\int f(\vec{x},\vec{v},t)d\vec{v}$
Vlasov equation
$\left(\frac{df}{dt}=\right)\frac{\partial f(\vec{x},\vec{v},t)}{\partial t}+\vec{v}\cdot\nabla_x f(\vec{x},\vec{v},t)+\frac{d\vec{v}}{dt}\cdot\nabla_v f(\vec{x},\vec{v},t)=0$
?
f
$\vec{x}\times\vec{v}\times t \to \mathbb{R}$
Initial condition
$f(\vec{x},\vec{v},0)=f_0(\vec{x},\vec{v})$

Figure 1.1.: Extended Model Pathway Diagram (MPD) of the Vlasov–Poisson system of equations [165]: Quantities are given in circles, the orange boxes denote the types of the quantities. Double circles denote the variables of the computational domain. Dashed black lines indicate constants and boundary conditions. Purple circles with question marks denote the unknown quantities that need to be solved for. The graph's cycle in the graph indicates that the unknown quantities are coupled and need to be solved for self-consistently.

This means that the electric and magnetic fields $\vec{E}$ and $\vec{B}$ need to be modeled self-consistently, for instance by the Maxwell or Poisson equations with the Vlasov equation. The Vlasov–Poisson system dependencies are illustrated in Figure 1.1 by an Extended Model Pathway Diagram (MPD)—a model visualization method developed in [93]. The distribution function f, the electric field $\vec{E}$ and the potential ϕ are the unknowns that the simulation needs to solve for. The cycle in the graph illustrates the property of self-consistency. To extend Figure 1.1 to the Vlasov–Maxwell system, one would additionally need to solve for $\vec{B}$ by adding the Maxwell equations to the MPD. As a consequence, great challenges lie in the modeling and numerical solution of the Vlasov system of equations.

This work is going to focus on resolving the Vlasov equation's distribution function on a phase-space grid, either in the full six-dimensional phase space, or on an appropriately dimensionally reduced phase space, such as with the gyrokinetic transform [14]. Later, we will look at two codes implementing full Vlasov with a Semi-Lagrangian approach (SeLaLib, cf. Section 3.5) and gyrokinetics with an Eulerian approach (GENE, cf. Section 5.3).

The Curse of Dimensionality

The main challenge of the grid-based simulation approach is the Curse of Dimensionality. If each of the d dimensions is resolved by N points, the number of degrees of freedom (DOF) is given by

$$\#\mathrm{DOF} = N^d \tag{1.2}$$

The phase-space resolution N should generally be high, to allow for the representation of filamentation phenomena of the Vlasov equation [54]. The resulting rapid growth of DOF through increasing the resolution at high dimensionalities d is known as the Curse of Dimensionality.

With dimensionalities d of four to six, the memory and compute requirements quickly become intractable for even today's most powerful supercomputers. For instance, storing the $6D$ distribution function f in double precision with 32 points per dimension already requires 8 GiB of main memory, which is currently a typical figure for laptops and workstation PCs. If the simulation needs to be run at twice the resolution, it will require at least 64 times the number of operations, and consume at least 64 times the memory: Then, at 512 GiB, the data is only going to fit onto the main memory of custom high-performance workstations (if one wants to use a single-node workstation system for the purpose). Doubling the resolution again, to 128 points per dimension, will already require at least 32 TiB, only for holding the distribution function in main memory. This is more than what quite a few of the current German Tier-3 HPC systems provide [81]. Thus, it is easy to see how the memory requirements for direct Vlasov simulations can quickly exhaust even the largest systems' resources.

1.2. Contributions in this Thesis

This work focuses on three key challenges for large-scale grid-based computations: ensuring numerical stability and conservation (Chapter 3), developing algorithms for scaling the simulation along the memory limit (Chapter 4), and making simulations on entire and multiple compute systems feasible (Chapter 5). Within a broader scope, the thesis attempts to advance some aspects related to different *Priority Research Directions* and *Cross-Cutting Research Directions* for Fusion Energy Research formulated by the US Department of Energy in its Exascale Requirements Review in 2017 [20]:

- Turbulence and Transport Simulations: High-resolution, high-fidelity simulations for 'Multiscale turbulence effects in plasma transport' [20, section 3.1.1].
- High-Energy-Density Laboratory Plasmas: 'High-dimensional partial differential equation (PDE) solvers (6D Vlasov)' [20, section 3.3.2].
- Radio-Frequent Heating and Current Drive: Predict how much power is coupled to the core plasma with 'full 6D linear delta-f type particle or continuum approach' [20, section 3.1.3].

The main method in this work is the sparse grid combination technique [61]: It allows re-using existing grid-based solvers by running multiple instances of the solver, each at a relatively low resolution. After certain time intervals, the *combination* step exchanges the information between the solvers, such that a drastic reduction of compute complexity can be achieved at a moderate cost in accuracy. As a foundation, an overview of sparse grid (SG) theory and the combination technique (CT) will be given in Chapter 2.

The numerical, algorithmic, and technical aspects are manifold and often connected: Basis functions need to be chosen, such that the stability and conservation of the sparse grid combination technique is ensured for the given partial differential equation (PDE) model. The CT code needs to be coupled to the simulation code, where both use shared and distributed memory parallelism. Since compute time and main memory are limited even on the biggest HPC systems, efficient algorithms and implementations are necessary to scale up to the desired problem sizes. At the same time, the code needs to be flexible enough to allow the comparison of different variants, for instance to compare different numerical solvers or basis functions. And finally, in order to connect HPC systems, it is necessary to explore the best ways of making them talk to each other despite the differences in hardware, software, and network security standards.

Addressing these questions comprises the core contributions of this thesis, which are summarized as follows:

1. A novel approach to the combination technique, which allows to replace the hierarchical hat function by other multiscale functions such as the mass-conserving CDF wavelets in a straightforward manner: This is particularly interesting, as numerical instability was a major roadblock for large-scale Vlasov simulations with the CT. The thesis shows in Chapter 3 that the standard hierarchical hat functions are the cause for the numerical instability, and that the mass-conserving CDF wavelets can be used to avoid it, while also obtaining better solution accuracy.

2. The extension of the open-source framework DisCoTec to scale up to the memory of entire supercomputing systems: Previous work had focused on strong scaling scenarios. Considering memory overheads of the parallelization, it is a challenge to scale up to the memory of an entire system by weak scaling, while a substantial fraction of the memory is still used for the simulations on the component grids. The thesis introduces various algorithmic and implementation improvements to DisCoTec in Chapter 4, that allow to scale up to the full system size of four German supercomputers, including the three CPU-based Tier-0/Tier-1 systems. The corresponding measurements are presented in Chapter 5.

3. Widely-distributed simulations, where two HPC systems synchronously run a single simulation: At extremely fine resolution scales, the memory of a single system can become insufficient to run a higher-dimensional CT simulation. In such cases, the inherent parallelism of the CT can be used to run the simulation on multiple systems at the same time. This is enabled by file transfer as well as sophisticated algorithms for assigning the different simulation instances to the systems, two of which are introduced in Chapter 4. The resulting drastic reduction in the communication volume was used to run joint simulations on different HPC systems synchronously, as evaluated in Chapter 5.

Parts of the work have already been published in:

- T. Pollinger et al. 'Leveraging the Compute Power of Two HPC Systems for Higher-Dimensional Grid-Based Simulations with the Widely-Distributed Sparse Grid Combination Technique'. In: SC '23. Association for Computing Machinery, Nov. 11, 2023. URL: `https://dl.acm.org/doi/10.1145/3581784.3607036` (visited on 11/15/2023) (Sections 4.5, 4.5.1, 4.5.3, 4.6 and 5.5)
- T. Pollinger et al. 'A Stable and Mass-Conserving Sparse Grid Combination Technique with Biorthogonal Hierarchical Basis Functions for Kinetic Simulations'. In: *Journal of Computational Physics* (July 7, 2023), p. 112338. URL: `https://www.sciencedirect.com/science/article/pii/S0021999123004333` (visited on 07/17/2023) (preprint at [136], Sections 2.1, 3.1, 3.2, 3.4 and 3.5)
- T. Pollinger et al. 'Distributing Higher-Dimensional Simulations Across Compute Systems: A Widely Distributed Combination Technique'. In: *2021 IEEE/ACM International Workshop on Hierarchical Parallelism for Exascale Computing (HiPar)*. 2021 IEEE/ACM International Workshop on Hierarchical Parallelism for Exascale Computing (HiPar). Nov. 2021, pp. 1–9 (Sections 4.5.1 and 4.5.2)

In addition to the aforementioned, the author contributed to the following published works, which could not be covered in this thesis:

- T. Pollinger, K. Kormann, and D. Pflüger. *Scaling the Plasma Simulation while Conserving the Mass: A Massively-Parallel Semi-Lagrangian Solver with the Sparse Grid Combination Technique*. Was awarded the Best Poster Award at PASC'22. PASC22, June 25, 2022. URL: `https://pasc22.pasc-conference.org/program/schedule/presentation/?id=pos109&sess=sess181` (visited on 07/19/2022)
- G. Daiß et al. 'Beyond Fork-Join: Integration of Performance Portable Kokkos Kernels with HPX'. in: *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). June 2021, pp. 377–386

- T. Pollinger and D. Pflüger. 'Learning-Based Load Balancing for Massively Parallel Simulations of Hot Fusion Plasmas'. In: *Advances in Parallel Computing* 36 (Parallel Computing: Technology Trends 2020), pp. 137–146. URL: `http://doi.org/10.3233/APC200034`
- R. Lago et al. 'EXAHD: A Massively Parallel Fault Tolerant Sparse Grid Approach for High-Dimensional Turbulent Plasma Simulations'. In: *Software for Exascale Computing - SPPEXA 2016-2019*. Lecture Notes in Computational Science and Engineering. Cham: Springer International Publishing, Jan. 1, 2020, pp. 301–329

Chapter 2

The Combination Technique: A Curse-Breaking Multiscale Method for Grid-Based Simulations

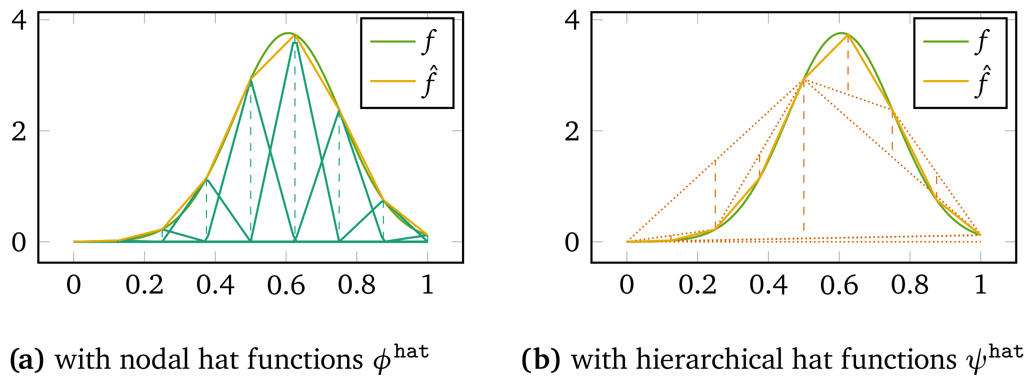
This chapter introduces the sparse grid (SG) combination technique (CT) and the concepts it is based on: Multiscale basis functions, and the SG subspace cut-off. The standard hierarchical hat functions will first be sketched from the 'usual' perspective taken in the SG literature (Section 2.1), and then viewed in higher detail, from the more general perspective of biorthogonal and lifting wavelets (Section 2.1.1). This general perspective allows one to use other basis functions, such as the biorthogonal basis functions and full weighting basis functions. Based on different multiscale basis functions, sparse grids and the combination technique are introduced in Sections 2.2 and 2.3. Some important extensions of the CT for time stepping are laid out in Section 2.3.2.

It is noted that the content of Sections 2.1.1 and 2.1.2 was developed for [137], in close collaboration with Johannes Rentrop, who fully contributed the formal analysis. The author of this thesis contributed to the conceptualization and visualization of the mass-conserving basis transformations, and carried out the pertaining experiments presented in Chapter 3.

2.1. Nodal and Hierarchical Function Space Bases

One of the most straightforward ways of numerically representing a one-dimensional function f on a closed interval $\Omega = [0,1]$ is the discretization on evenly-spaced collocation points, and linear interpolation in between.

On this space of piecewise linear functions V, a function can be expressed as the weighted sum of linear basis functions $\phi^{\mathtt{hat}}$ (a.k.a. triangular, triangle, hat, or tent functions), cf. Figure 2.1a. The weight coefficients that the $\phi^{\mathtt{hat}}$ are multiplied by will generally be proportional to the function values.



(a) with nodal hat functions $\phi^{\mathtt{hat}}$ **(b)** with hierarchical hat functions $\psi^{\mathtt{hat}}$

Figure 2.1.: Linear interpolation $\hat{f}$ of a Gaussian function $f = \mathcal{N}(\mu = \frac{1}{2} + \frac{1}{3\pi}, \sigma = 0.15)$ at resolution level $\ell = 3$: The length of the vertical lines denotes the value of the basis function coefficients. Note that the resulting interpolant $\hat{f}$ is the same in both the nodal and hierarchical/multiscale representations.

As we go finer and finer in resolution—say we insert collocation points in between the existing ones for each refinement level ℓ—we get a hierarchy of nested 'nodal' function spaces

$$V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_\ell \subset \cdots \subset L^2(\mathbb{R}). \tag{2.1}$$

Then, the number of collocation points at locations $i \cdot h, i = 0, 1, \ldots, h = 2^{-\ell}$ is given by $N \approx 2^\ell$. With collocation points on both sides of the interval, $N = 2^\ell + 1$, and for one-sided boundaries, such as with periodic boundary conditions, $N = 2^\ell$. Accordingly, the basis functions can be explicitly given as

$$\phi^{\mathtt{hat}}_{i/\ell}(x) = \max\left(1 - \left|x - i \cdot 2^{-\ell}\right|, 0\right) , \quad i = 0, 1, \ldots, N \, . \tag{2.2}$$

A function f will be approximated in V_ℓ with a resolution of h.

As an alternative to the nodal function space V_ℓ on level ℓ, one can use a union of 'hierarchical increment spaces' W_ℓ, which are the complement of $V_{\ell-1}$ in V_ℓ

$$V_\ell = V_{\ell-1} \oplus W_\ell \ , \qquad V_{\ell-1} \cap W_\ell = \{0\} \tag{2.3}$$

(with $W_0 = V_0$ for simplicity). The standard choice of basis functions ψ for representing W is to use hat functions as well [15], see Figure 2.1b. This time, they are understood as a hierarchical basis function $\psi^{\mathtt{hat}}$: The nodal and hierarchical bases contain the same information—piecewise linear functions:

$$\hat{f} = \sum_{i=0}^{N-1} c_i \phi_i^{\mathtt{hat}} = \sum_{i=0}^{N-1} \alpha_i \psi_i^{\mathtt{hat}} \ . \tag{2.4}$$

In contrast to the nodal representation, the hierarchical representation is a multiscale representation: As Figure 2.1b shows, the support of the hierarchical hat functions starts with all of Ω on levels 0 and 1 and gets smaller as the level ℓ increases. While the basis coefficients c_i will be proportional to the function value of f at the respective position, the α_i denote how much the fine resolution features deviate from the larger-resolution features. This comes at the cost of losing some locality of the multiscale basis functions' support on the coarser levels, where the coarser scales reside. For this reason, the hierarchical basis coefficients are often termed 'hierarchical surplusses'. The transformation from the nodal to hierarchical basis representation (and corresponding coefficients) is called *hierarchization*, the inverse transformation is called *dehierarchization* [63].

2.1.1. Biorthogonal Wavelets as Hierarchical Bases (and Vice Versa)

The concept of nested nodal function spaces V and their increment spaces W outlined with Equations (2.1) and (2.3) is exactly the same in the broader scope of (biorthogonal) wavelet theory—although the function space does not necessarily consist of piecewise linear functions. What changes, however, is that both nodal functions ϕ and multiscale functions ψ are entirely free to choose: They are not required to be interpolating or have the same shape, nor do they even need to be continuous.

Instead, the functions need to fulfill a set of conditions:

1. Dilates and translates of ϕ form a basis for V_ℓ:

$$\phi_{\ell,s}(x) := \phi(2^\ell x - s) \ , \quad V_\ell = \mathrm{span}\{\phi_{\ell,s}\}_{s\in\mathbb{Z}} \ . \tag{2.5}$$

2. ϕ fulfills the two-scale equation with coefficients $h_s \in \mathbb{R}$:

$$\phi(x) = \sum_{s\in\mathbb{Z}} h_s \phi(2x - s). \tag{2.6}$$

The two-scale equation gives ϕ its name as the *scaling function*.

3. Dilates and translates of ψ form a basis for the complement space on each level ℓ

$$\psi_{\ell,s}(x) := \psi(2^{\ell-1}x - s)\ , \quad W_\ell = \operatorname{span}\{\psi_{\ell,s}\}_{s\in\mathbb{Z}}, \tag{2.7}$$

in particular $\psi_{1,0} = \psi$.

These requirements have two implications.

First, by Equation (2.3), every function $f \in L^2$ can be represented by both sets of basis functions:

$$f = \sum_{\ell=0}^{\infty}\sum_{s\in\mathbb{Z}} c_{\ell,s}\phi_{\ell,s}\ , = \sum_{\ell=0}^{\infty}\sum_{s\in\mathbb{Z}} \alpha_{\ell,s}\psi_{\ell,s}. \tag{2.8}$$

(Note that this is a generalization of Equation (2.3).) These hierarchical functions $\psi_{\ell,s}$ are called *wavelets*, where ψ is the mother wavelet, and the α are *wavelet coefficients*. Hybrid representations with both wavelet and scaling function coefficients are possible [137].

Second, there are coefficients $g_s \in \mathbb{R}$ to construct the mother wavelet ψ from the scaling function:

$$\psi(x) = \sum_{s\in\mathbb{Z}} g_s \phi(2x - s)\ . \tag{2.9}$$

Choosing the complement space W_ℓ to be exactly the orthogonal complement of $V_{\ell-1}$ leads to orthonormal wavelet bases [30]. *Biorthogonal* wavelets, by contrast, require a set of dual scaling functions $\tilde{\phi}$ and dual wavelets $\tilde{\psi}$, as well as coefficients $\tilde{h}_s, \tilde{g}_s$ that fulfill

$$\tilde{\phi}(x) = 2\sum_{s\in\mathbb{Z}} \tilde{h}_s \tilde{\phi}(2x - s)\ , \qquad \tilde{\psi}(x) = \sum_{s\in\mathbb{Z}} \tilde{g}_s \tilde{\phi}(2x - s)\ . \tag{2.10}$$

If $\int \phi(x)\tilde{\phi}(x-s)\mathrm{d}x = \delta_{s0}$, then the following biorthogonality relations (using the L^2 scalar product $\langle\cdot,\cdot\rangle$) hold [25]:

$$\begin{aligned} \langle \phi_{\ell,s}, \tilde{\phi}_{\ell,s'}\rangle &= 2^{-\ell}\delta_{ss'} \,, \qquad & \langle \psi_{\ell,s}, \tilde{\psi}_{\ell',s'}\rangle &= 2^{-\ell}\delta_{ll'}\delta_{ss'} \,, \\ \langle \phi_{(\ell-1)s}, \tilde{\psi}_{\ell,s'}\rangle &= 0 \,, & \langle \psi_{\ell,s}, \tilde{\phi}_{(\ell-1)s'}\rangle &= 0 \,. \end{aligned} \tag{2.11}$$

Finite numbers of nonzero coefficients $h_s, \tilde{h}_s, g_s, \tilde{g}_s$ can be obtained [25] by setting them as

$$g_s := (-1)^{1-s}\tilde{h}_{1-s} \,, \qquad \tilde{g}_s := (-1)^{1-s}h_{1-s} \tag{2.12}$$

for a choice of h_s and $\tilde{h}_s$ that also fulfill Equations (2.5) to (2.7). (The original paper by Cohen, Daubechies, and Feauveau [25] lists a variety of suitable coefficients for B-spline wavelets.)

Then, the wavelet coefficients (or hierarchical surplusses) α for the original wavelet ψ are given by the scalar product with the dual wavelet $\tilde{\psi}$, and vice versa:

$$f = \sum_{\ell=0}^{\infty}\sum_{s\in\mathbb{Z}} 2^{\ell}\langle \tilde{\psi}_{\ell,s}, f\rangle \psi_{\ell,s} = \sum_{\ell=0}^{\infty}\sum_{s\in\mathbb{Z}} 2^{\ell}\langle \psi_{\ell,s}, f\rangle \tilde{\psi}_{\ell,s} \,. \tag{2.13}$$

Luckily, one does not need to explicitly compute integrals to perform the hierarchization or *wavelet transform*. In fact, one does not even need to know the scaling function ϕ or the wavelet ψ explicitly, it suffices to know the coefficients h_s and $\tilde{h}_s$.

Suppose a function $f_\ell \in V_\ell$ is given in the scaling function representation $f_\ell = \sum_{s\in\mathbb{Z}} c_{\ell,s}\phi_{\ell,s}$. Then, the wavelet coefficients $\alpha_{\ell,k}$ on level ℓ are determined by

$$\begin{aligned} \alpha_{\ell,k} &= 2^{\ell}\langle \tilde{\psi}_{\ell,k}, f_l\rangle = 2^{\ell}\sum_{s\in\mathbb{Z}} c_{\ell,s}\langle \tilde{\psi}_{\ell,k}, \phi_{\ell,s}\rangle = \sum_{s\in\mathbb{Z}}\sum_{s'\in\mathbb{Z}} c_{\ell,s}\tilde{g}_{s'}2^{\ell}\langle \tilde{\phi}_{\ell,(s'+2k)}, \phi_{\ell,s}\rangle \\ &= \sum_{s\in\mathbb{Z}}\sum_{s'\in\mathbb{Z}} c_{\ell,s}\tilde{g}_{s'}\delta_{(s'+2k)s} = \sum_{s\in\mathbb{Z}} c_{\ell,s}\tilde{g}_{(s-2k)}, \end{aligned} \tag{2.14}$$

and the coarser scaling function coefficients $c_{(\ell-1)k}$ can be computed as

$$\begin{aligned} c_{(\ell-1)k} &= 2^{\ell-1}\langle \tilde{\phi}_{(\ell-1)k}, f_l\rangle = 2^{\ell-1}\sum_{s\in\mathbb{Z}} c_{\ell,s}\langle \tilde{\phi}_{(\ell-1)k}, \phi_{\ell,s}\rangle \\ &= \sum_{s\in\mathbb{Z}}\sum_{s'\in\mathbb{Z}} c_{\ell,s}\tilde{h}_{s'}2^{\ell}\langle \tilde{\phi}_{\ell(s'+2k)}, \phi_{\ell,s}\rangle = \sum_{s\in\mathbb{Z}} c_{\ell,s}\tilde{h}_{(s-2k)} \,. \end{aligned} \tag{2.15}$$

This hierarchization procedure can be performed recursively for decreasing ℓ.

Conversely, the dehierarchization–now called *inverse wavelet transform*–recovers the original scaling function coefficients

$$\begin{aligned} c_{\ell,k} = 2^{\ell}\langle\tilde{\phi}_{\ell,k}, f_l\rangle &= \sum_{s\in\mathbb{Z}} c_{(\ell-1)s} 2^{\ell}\langle\tilde{\phi}_{\ell,k}, \phi_{(\ell-1),s}\rangle + \sum_{s\in\mathbb{Z}} \alpha_{\ell,s} 2^{\ell}\langle\tilde{\phi}_{\ell,k}, \psi_{\ell,s}\rangle \\ &= \sum_{s\in\mathbb{Z}} c_{(\ell-1)s} h_{k-2s} + \sum_{s\in\mathbb{Z}} \alpha_{\ell,s} g_{k-2s} \ , \end{aligned} \tag{2.16}$$

again recursively for increasing ℓ.

All of these observations are also entirely valid for higher-order continuous biorthogonal wavelets [25] and by allowing discontinuities, it is even possible to recover the orthogonality for compactly supported (multi)wavelets [3]. Furthermore, different nesting sequences of the piecewise linear functions in V can be beneficial for numerical approximations of functions with certain properties. Examples include the Clenshaw-Curtis/Chebychev [118], Gauss-Lobatto, and Leja [116] points. However, for the purposes of this thesis we always assume spaces of equidistant piecewise linear continuous functions V and W on a periodic domain Ω. In particular, three possible choices for the hierarchical multiscale basis are considered. It is noted that the lifting scheme [154, p. 14.3] can be used to transform these equidistant discretizations to other nested function representations.

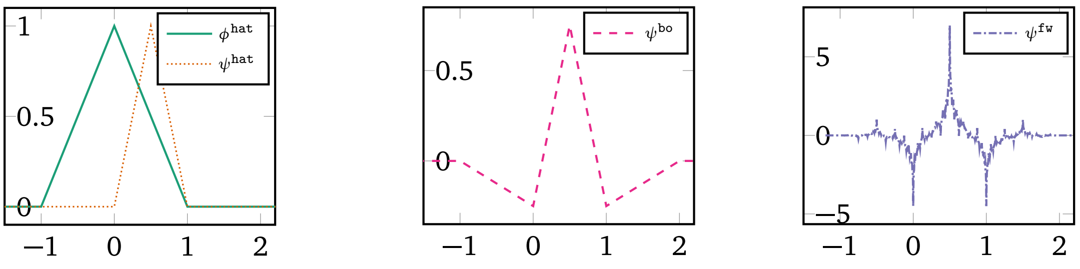
Hierarchical Hat Function in the Biorthogonal Framework

For most of the sparse grid literature, the hierarchical hat function serves as the standard choice for the multiscale basis function. Its scaling function $\phi^{\mathtt{hat}}$ is also a hat function, which means that function values and scaling function coefficients are equivalent, and it is an example of a generalized hierarchical basis [97]. Sweldens [154] noted that $\psi^{\mathtt{hat}}$ denotes a special case of biorthogonal wavelets, as its 'dual wavelet' is the Dirac delta distribution. This leads to beneficial properties in the hierarchical hat function, which hold for all generalized hierarchical bases [97]: First, the basis transforms are extremely efficient ('lazy wavelets' [154]). Second, due to the interpolating property of the hierarchical hat function, the finer-scale wavelet coefficients do not influence function values / scaling function coefficients on coarser scales. As a consequence, the wavelet transform can not only be computed from the finest to the coarsest level, but each wavelet coefficient can be immediately computed for any point if the function values of its two coarser-scale neighbors are known (the 'aunt' property [154]). This is a particularly valuable feature for adaptive methods, such as optimization [162] and uncertainty quantification [143]—when more and more points should be added without affecting values at the previous points—and also for collocation-based solvers [4].

The hierarchical hat function is defined by the filter coefficients

$$\begin{aligned} (h_s)^{\texttt{hat}}_{-1\le s\le 1} &= (\tfrac{1}{2}, 1, \tfrac{1}{2})\,, \quad (g_s)^{\texttt{hat}}_{0\le s\le 2} = (0,1,0)\,, \\ (\tilde{h}_s)^{\texttt{hat}}_{-1\le s\le 1} &= (0,1,0)\,, \quad (\tilde{g}_s)^{\texttt{hat}}_{0\le s\le 2} = (-\tfrac{1}{2}, 1, -\tfrac{1}{2})\,. \end{aligned} \tag{2.17}$$

It is a special case of hierarchical B-spline bases [162] (first order), and of the interpolet basis [33, 97] (also first order). In the context of geometric multigrid methods, the level-wise wavelet transform corresponds to the injection restriction operator [67] for the scaling function coefficients. Both the hierarchical hat function $\psi^{\texttt{hat}}$ and the 'hat' scaling function $\phi^{\texttt{hat}}$ are depicted in Figure 2.2a. Recall also the illustration of the two different ways of interpolating a function given in Figure 2.1.



(a) hierarchical hat $\psi^{\texttt{hat}}$ **(b)** biorthogonal $\psi^{\texttt{bo}}$ **(c)** full weighting $\psi^{\texttt{fw}}$

Figure 2.2.: Different hierarchical / multiscale functions ψ to span the linear increment spaces $W_{\ell'}$. Note that $\psi^{\texttt{bo}}$ was defined in the same way in [15, p. 64], but plotted wrongly in Figure 4.12 therein.

Mass-Conserving Hierarchical Basis Functions: The 'Biorthogonal' and 'Full Weighting' Functions

One of the initial motivations to viewing sparse grids from the perspective of biorthogonal wavelets was the possibility of introducing conservation of mass, which can be a desirable property as will be detailed in Chapter 3. To achieve conservation of mass, one needs to allow for a slightly higher number of nonzero coefficients in h_s and $\tilde{h}_s$ (see also Section 2.1.2). The next possible choice for a linear ansatz space V_ℓ are the dual functions ${}_{2,2}\psi$ and ${}_{2,2}\tilde{\psi}$ in [25], which are termed *biorthogonal* $\psi^{\texttt{bo}}$ and *full weighting* $\psi^{\texttt{fw}}$ functions in this thesis. 'Full weighting' refers to the fact that the wavelet transform modifies the scaling function coefficients exactly the same way as the full weighting restriction operator in geometric multigrid methods [67]. Furthermore, it is used for lossless compression in the JPEG 2000 standard, where it is referred to as the 'CDF 5/3' wavelet.

The biorthogonal basis function $\psi^{\mathtt{bo}}$ is defined by

$$\begin{aligned}(h_s)^{\mathtt{bo}}_{-1\le s\le 1} &= (\tfrac{1}{2}, 1, \tfrac{1}{2})\,, & (g_s)^{\mathtt{bo}}_{-1\le s\le 3} &= (-\tfrac{1}{8}, -\tfrac{1}{4}, \tfrac{3}{4}, -\tfrac{1}{4}, -\tfrac{1}{8})\,,\\ (\tilde{h}_s)^{\mathtt{bo}}_{-2\le s\le 2} &= (-\tfrac{1}{8}, \tfrac{1}{4}, \tfrac{3}{4}, \tfrac{1}{4}, -\tfrac{1}{8})\,, & (\tilde{g}_s)^{\mathtt{bo}}_{0\le s\le 2} &= (-\tfrac{1}{2}, 1, -\tfrac{1}{2})\,,\end{aligned} \tag{2.18}$$

and the full weighting basis function $\psi^{\mathtt{fw}}$ by

$$\begin{aligned}(h_s)^{\mathtt{fw}}_{-2\le s\le 2} &= (-\tfrac{1}{4}, \tfrac{1}{2}, \tfrac{3}{2}, \tfrac{1}{2}, -\tfrac{1}{4})\,, & (g_s)^{\mathtt{fw}}_{0\le s\le 2} &= (-\tfrac{1}{2}, 1, -\tfrac{1}{2})\,,\\ (\tilde{h}_s)^{\mathtt{fw}}_{-1\le s\le 1} &= (\tfrac{1}{4}, \tfrac{1}{2}, \tfrac{1}{4})\,, & (\tilde{g}_s)^{\mathtt{fw}}_{-1\le s\le 3} &= (-\tfrac{1}{8}, -\tfrac{1}{4}, \tfrac{3}{4}, -\tfrac{1}{4}, -\tfrac{1}{8})\,.\end{aligned} \tag{2.19}$$

From the shape of the multiscale functions in Figures 2.2b and 2.2c, one can see why they conserve mass: The integral over ψ is equal to 0, which means that during dehierarchization, anything that is added on the finer scales will not contribute any new mass, but 'shift around' the mass that was already present on the coarser level. This is equivalent to the observation that the sum of the filter coefficients g and $\tilde{g}$ is equal to 0.

Another desirable property of both these functions is that they can be considered lifting wavelets [154], and can therefore be computed in-place with two data sweeps per level—it is not necessary to copy the data to apply the hierarchization and dehierarchization operations (which holds also true for $\psi^{\mathtt{hat}}$ if the transforms are recursed in the 'usual' directions).

Note that different normalization factors compared to Cohen, Daubechies, and Feauveau [25] were chosen, which avoids unnecessary arithmetic operations in the basis transforms.

2.1.2. Comparison of Mass-Conserving and Standard Hat Functions' Theoretical Properties

All multiscale basis functions—$\psi^{\mathtt{hat}}$, $\psi^{\mathtt{bo}}$, and $\psi^{\mathtt{fw}}$—allow for representing the hierarchical increment spaces $W_{\ell'}$. The hierarchization and dehierarchization operations can be performed in-place and in linear complexity on each level [137]. In fact, the wider-supported basis functions $\psi^{\mathtt{bo}}$ and $\psi^{\mathtt{fw}}$ can be constructed from $\psi^{\mathtt{hat}}$ by a lifting scheme [154].

The benefits of $\psi^{\mathtt{hat}}$ were briefly discussed in Section 2.1.1 and unfortunately, the interpolating property is lost for the mass-conserving two functions. For the purposes of solving PDEs however, the crucial difference between $\psi^{\mathtt{hat}}$, $\psi^{\mathtt{bo}}$, and $\psi^{\mathtt{fw}}$ is that the latter two provide stability as well as conservation of mass, while the former does not. The reason is that the dual 'wavelet' for the hierarchical hat function is the Dirac delta function, which has poor regularity (it is not even in L_2) [154]. For biorthogonal wavelets, the regularity, stability, and conservation properties are closely connected [25, 26, 97].

Two implications, which are justified in more detail in [137], are as follows:

1. The number of vanishing moments of the *primal* wavelet is equal to the polynomial exactness of the *dual* scaling function and vice versa [25]. Since the Dirac delta distribution has no degree of polynomial exactness, there is no conservation of moments (in particular, mass) between the different hierarchization levels of $\psi^{\mathtt{hat}}$. By contrast, $\psi^{\mathtt{bo}}$ and $\psi^{\mathtt{fw}}$ conserve the mass and momentum but no higher-order moments, as the polynomial exactness of the respective dual scaling functions is two.

2. The *decay* of the wavelet coefficients characterizes the Sobolev regularity of a function. If the Sobolev norm is finite, then the wavelet coefficients necessarily decay. In particular, the Sobolev regularity of the hierarchical hat function is too low to assert L_2 or even H_1 stability of the multiscale basis. For the biorthogonal and full weighting functions, L_2 stability can be proven, which leads to H^r stability in the multivariate functions [137].

2.2. Sparse Grids

Sparse grids (SGs) [15, 173] are constructed from multivariate multiscale functions. To this end, one can represent a d-dimensional function in $V_{\vec{\ell}}$ with either nodal or increment spaces through tensor products of the respective one-dimensional spaces:

$$V_{\vec{\ell}} = V_{\ell_1} \otimes V_{\ell_2} \otimes \ldots \otimes V_{\ell_d} \tag{2.20}$$

$$W_{\vec{\ell}'} = W_{\ell'_1} \otimes W_{\ell'_2} \otimes \ldots \otimes W_{\ell'_d} \tag{2.21}$$

$$V_{\vec{\ell}} = \bigoplus_{\vec{\ell}'=\vec{0}}^{\vec{\ell}} W_{\vec{\ell}'}. \tag{2.22}$$

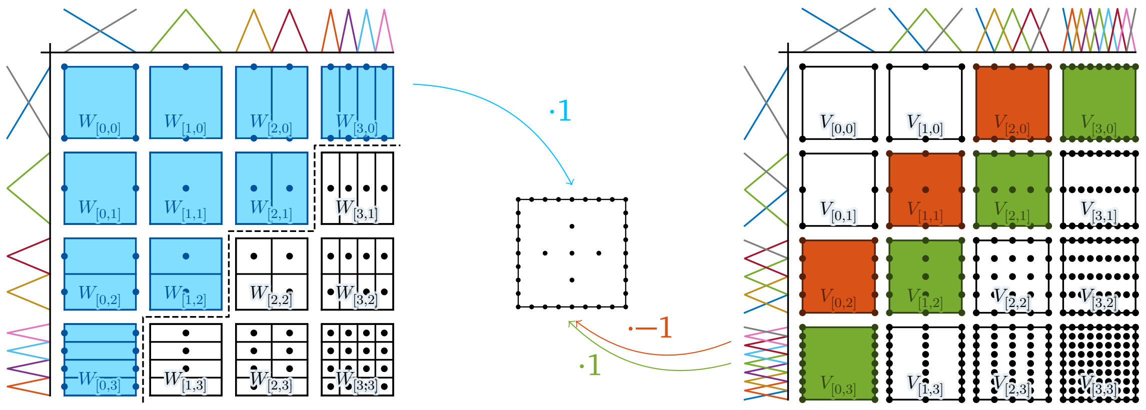


Figure 2.3.: SGs with hierarchical increment spaces vs. the combination technique (CT)—the dots denote the location of collocation points / DOF in the case of (hierarchical) hat functions with boundaries on both sides.
Graphics based on scripts adapted from [162, 163], Copyright © 2019 Julian Valentin, licensed under CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0/).

The central sparse grid idea states that not all possible d-dimensional tensor products $W_{\vec{\ell}'}$ of the hierarchical spaces $W_{\ell'}$ are necessary for the good approximation of a function. Instead, only those $W_{\vec{\ell}'}$ that contribute most to the function (typically measured in the L_2 norm) are used within the sparse grid. For all selected $W_{\vec{\ell}'}$, the set of level vectors $\{\vec{\ell}_i'\}$ is termed the *index set* $\mathcal{I}^{\text{SG}}$ of the sparse grid.

The 'standard' approach selects $\mathcal{I}^{\text{SG}}$ to be a simplex subset in the level space of $W_{\vec{\ell}'}$ to form the sparse grid of target resolution $\vec{\ell} = (L, \ldots, L), L \in \mathbb{N}_0$:

$$\mathcal{I}^{\text{SG}} = \left\{ \vec{\ell}' \in \mathbb{N}_0^d : \left| \vec{\ell}' \right|_1 \leq L \right\} . \tag{2.23}$$

As displayed left in Figure 2.3, this means that $\mathcal{I}^{\text{SG}}$ is a (light blue) triangle in the level space for $d = 2$, and results in the typical sparse-grid structure (middle of the figure).

However, the choice of $\mathcal{I}^{\text{SG}}$ is only restricted by the requirement that it should be downward closed, or

$$\forall \vec{\ell} \in \mathcal{I}^{\text{SG}} : \vec{\ell}' \leq \vec{\ell} \implies \vec{\ell}' \in \mathcal{I}^{\text{SG}}, \tag{2.24}$$

which allows for different types of adaptivity [53, 125]. In particular, one can introduce a minimum level $\vec{\ell}^{\min}$, up to which the full grid is contained in the sparse grid, similar to 'sparse grids with coarser boundaries' [162, Sec. 2.4.1]. This leads to the generalized hybrid sparse grid representation of f

$$f^{\mathrm{SG}}_{\vec{\ell}^{\min}}(\vec{x}) = \sum_{\vec{s}\in\mathbb{Z}^d} c_{\vec{\ell}^{\min},\vec{s}}\,\phi_{\vec{\ell}^{\min},\vec{s}}(\vec{x}) + \sum_{\vec{\ell}'\in\mathcal{I}^{\mathrm{SG}}} \sum_{\vec{s}\in\mathbb{Z}^d} \alpha_{\vec{\ell}',\vec{s}}\,\psi_{\vec{\ell}',\vec{s}}(\vec{x}) \,. \tag{2.25}$$

If the level range L is chosen uniformly, f is approximated on a target level $\vec{\ell}^{\max} := \vec{\ell}^{\min} + \vec{1}\cdot L$.

With the standard choice of $\mathcal{I}^{\mathrm{SG}}$ (2.23), i. e., $\vec{\ell}^{\min} = \vec{0}$ in Equation (2.25), one can show that the number of DOF of the sparse grid is of order $\mathcal{O}(2^L(\log 2^L)^{d-1})$ [15, 51], which is dramatically less than the $\mathcal{O}(2^{Ld})$ DOF of the full tensor product grid on $\vec{\ell}^{\max}$.

At the same time, the sparse grid interpolation error increases only slightly (if the function $f(\vec{x})$ is smooth enough), by a factor of $\mathcal{O}(\log(2^L)^{d-1})$ for the hierarchical hat function [92, Eq. (3.16) with $T = 0$]. For the mass-conserving basis functions, the interpolation error can be of the same order as in the full grid case, if $f(\vec{x})$ is less regular [137].

2.3. Sparse Grid Combination Technique

The sparse grid combination technique [61] is another way of getting in to the sparse grid space and is closely connected to Smolyak quadrature [152] (if dyadic refinement is assumed). In the combination technique (CT), many anisotropically resolved full grid functions $f^{\mathrm{FG},\vec{\ell}}$ living in the nodal spaces $V_{\vec{\ell}}$ are added up with different combination coefficients $c^c_{\vec{\ell}}$

$$f^{\mathrm{CT}}(\vec{x}) = \sum_{\vec{\ell}\in\mathcal{I}^{\mathrm{CT}}} c^c_{\vec{\ell}} \cdot f^{\mathrm{FG},\vec{\ell}}(\vec{x}) \tag{2.26}$$

to form a sparse grid, cf. the right side of Figure 2.3. These individual full grids are called *component grids*.

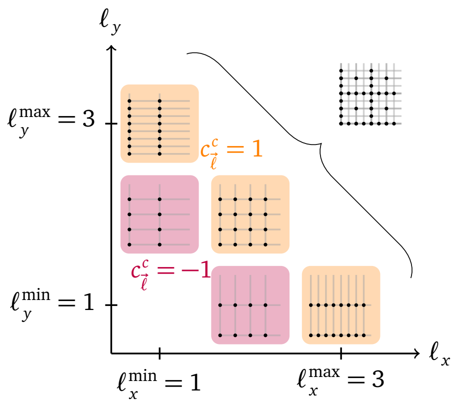
For a general index set $\mathcal{I}$, the combination coefficients are given by

$$c^c_{\vec{\ell}} = \sum_{\vec{z}=\vec{0}}^{\vec{1}} (-1)^{|\vec{z}|_1} \chi^{\mathcal{I}^{\mathrm{SG}}}(\vec{\ell}+\vec{z}) \,. \tag{2.27}$$

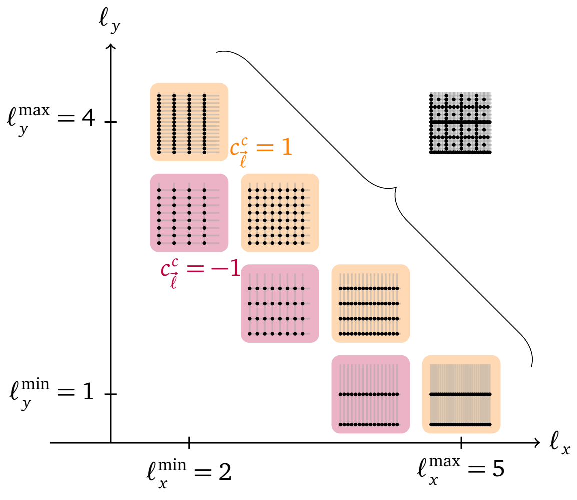
Here, χ denotes the indicator function of the sparse grid index set $\mathcal{I}^{\text{SG}}$ [71], and the levels where the coefficients are nonzero are part of the combination technique index set $\mathcal{I}^{\text{CT}}$. They are the levels of the component grids living in $V_{\vec{\ell}}$, on which f needs to be evaluated. Equation (2.27) can be interpreted as a multidimensional inclusion-exclusion principle, if one considers that each of the full grid functions $f^{\text{FG},\vec{\ell}}$ lives in the nodal tensor product space $V_{\vec{\ell}} = \oplus_{\vec{\ell}'=\vec{0}}^{\vec{\ell}} W_{\vec{\ell}'}$, cf. Equation (2.3). Thus, by building a sparse grid from full grids, one has to account for those hierarchical increment spaces $W_{\vec{\ell}'}$ that were 'counted' multiple times, by subtracting lower-resolved full grids that contain them. For the 'standard' sparse grid index set (2.23), the combination formula simplifies to

$$f^{\text{CT}}(\vec{x}) = \sum_{q=0}^{d-1}(-1)^q \binom{d-1}{q} \cdot \sum_{\left|\vec{\ell}\right|_1 = L-q} f^{\text{FG},\vec{\ell}}(\vec{x}). \tag{2.28}$$

In this case, the combination technique index set $\mathcal{I}^{\text{CT}}$ is a d-layered simplex in the level vector space, where the 'highest' layer with level sum $\left|\vec{\ell}\right|_1 = L$ is called the *main diagonal*. The blue expression denotes c_q^c [51], the coefficient pertaining to the qth diagonal in $\mathcal{I}^{\text{CT}}$. In two dimensions, the c_q^c are 1 and -1, see also Figure 2.3.



(a) with $\vec{\ell}^{\min} = (1,1)$, $\vec{\ell}^{\max} = (3,3)$



(b) with $\vec{\ell}^{\min} = (2,1)$, $\vec{\ell}^{\max} = (5,4)$

Figure 2.4.: Two different two-dimensional combination schemes with the regular truncated sparse grid combination technique, using periodic boundaries.

If a minimum level $\vec{\ell}^{\min} \neq \vec{0}$ is introduced like in Equation (2.25), this results in the truncated CT [10], where the combination technique index set is offset by $\vec{\ell}^{\min}$ compared to Equation (2.28), cf. Figure 2.4.

When solving a PDE, an obvious advantage of the CT over the SG discretization is that the differential operators need not be represented in the multiscale basis. Instead, existing solvers that operate on (full) structured grids can be readily used on each component grid, and the solver steps can be executed embarrassingly parallel between the component grids. Such solvers may require a minimum resolution for appropriate parallelization or to ensure stability in their solutions, which motivates the use of a minimum level.

Note that, compared to the sparse grid discretization (2.25), the CT duplicates DOF between component grids, such that the number of DOF is given by $\mathcal{O}((d \cdot 2^L(\log 2^L)^{d-1}))$ [59]. The interpolation error estimates are the same as for the sparse grid discretization, as CT and SG can represent the same function space. However, different operations like solving a PDE may result in very different functions between the (hierarchical) sparse grid and (nodal) CT discretizations [61] and exact error bounds can only be derived if the solver's errors are taken into account.

2.3.1. Error Cancellation in the Combination Technique

A quite intuitive concept of the errors in the CT is offered by the error cancellation property, based on an *error splitting assumption* [119]. The assumption states that the decomposition coefficients C of the pointwise error

$$
\begin{aligned}
\left|u^{\text{exact}}(\vec{x}) - u^{\text{CT}}(\vec{x})\right| = &\sum_{k=1}^{d} C_k(h_{\ell_k}) f_1(h_{\ell_k}) \\
&+ \sum_{k=1}^{d} \sum_{\substack{j=1 \\ j\neq k}}^{d} C_{k,j}(h_{\ell_k}, h_{\ell_j}) f_2(h_{\ell_k}, h_{\ell_j}) \\
&+ \ldots \\
&+ C_{1,\ldots,d}(h_{\ell_1}, \ldots, h_{\ell_d}) f_d(h_{\ell_1}, \ldots, h_{\ell_d})
\end{aligned}
\tag{2.29}
$$

can be bounded by a positive constant κ for the method at hand (interpolation, measurement, PDE solver, ...) for a given $\vec{x}$. The mesh width is $h_{\ell_k} = 2^{-\ell_k}$ in dimension k, and the functions f_i denote the respective error decay / convergence rate. It should approach zero as the $h_{\ell_k} \to 0$. This poses a relatively weak assumption, which typical PDE solver schemes will strive to fulfill by default.

If the assumption holds, then the multidimensional telescoping sum of the CT formula (2.27) leads to a cancellation of ‘under-resolution’ errors on the different component grids. The remaining error depends on how fast the mixed discretization errors decay, but the axis-aligned errors will always be cancelled [119, Section 2.3.1]. A quick decay is beneficial to obtain low errors in the combined solution, very similar to dimensionality reduction methods based on ANOVA [147]. This property is the reason why it can be sensible to rotate the coordinate axes within the domain [145] in a way that most of the variance is captured along the main axes.

One can conclude that the CT is aptly suited to break the curse of dimensionality for grid-based solvers. It provides beneficial practical properties compared to SG solvers, most notably that any existing solver operating on structured grids can be used to solve the PDE problem.

2.3.2. Combination Technique for Time-Dependent Problems

There are various modifications of the CT, some of which are utilized in this work. The most important variations to this work are shortly reviewed: The time-stepping CT that synchronizes the individual component grids, the adaptive CT that fits combination schemes to the problem at hand, and lastly the asynchronous time-stepping CT that allows to overlap the combination with the solver times.

Time Stepping: Recombination and Decombination

For time-dependent problems such as Equation (1.1), an arbitrary structured-grid solver can update the solution on each component grid for a given simulation time interval, after which the solutions are synchronized (‘combined’). This interval is called the *combination interval*, and it may contain multiple solver time steps. Algorithm 2.1 gives a high-level overview of the entire simulation with the CT.

Typically, the combination step contains two basis changes, the hierarchization and the dehierarchization, i. e., from the nodal representation to the hierarchical representation and back. While the full grid data is in the hierarchical representation, the coefficients need to be reduced to the sparse grid, and then scattered back to the component grids. Section 4.4 is going to discuss different variants of this reduction operation in more detail.

Algorithm 2.1 Time-Stepping CT

```
1:  procedure TimeSteppingCT($\mathcal{I}^{\text{CT}}$, $\mathcal{I}^{\text{SG}}$)
2:      for $t \leftarrow 0, N_{\text{steps}}$ do
                                                                              run
3:          for all $\vec{\ell} \in \mathcal{I}^{\text{CT}}$ do
4:              solve $\vec{\ell}$          ▷ Update component grids from time step $t$ to $t+1$
5:          end for
                                                                          combine
                                                                    Recombination
6:          for all $\vec{\ell} \in \mathcal{I}^{\text{CT}}$ do
7:              hierarchize $\vec{\ell}$                ▷ Basis transform: nodal → hierarchical
8:          end for
9:          Reduce($\mathcal{I}^{\text{CT}}$, $\mathcal{I}^{\text{SG}}$)
              ▷ Reduce hierarchical coefficients to sparse grid with Equation (2.26)
                                                                    Decombination
10:         Scatter($\mathcal{I}^{\text{CT}}$, $\mathcal{I}^{\text{SG}}$) ▷ Scatter hierarchical coefficients to component grids
11:         for all $\vec{\ell} \in \mathcal{I}^{\text{CT}}$ do
12:             dehierarchize $\vec{\ell}$              ▷ Basis transform: hierarchical → nodal
13:         end for
14:     end for
15: end procedure
```

This thesis calls the first part, consisting of hierarchization and reduction, the *(re)combination* step, and the second part, consisting of scatter and dehierarchization, the *decombination* step. The existing literature typically summarizes everything as 'recombination'.

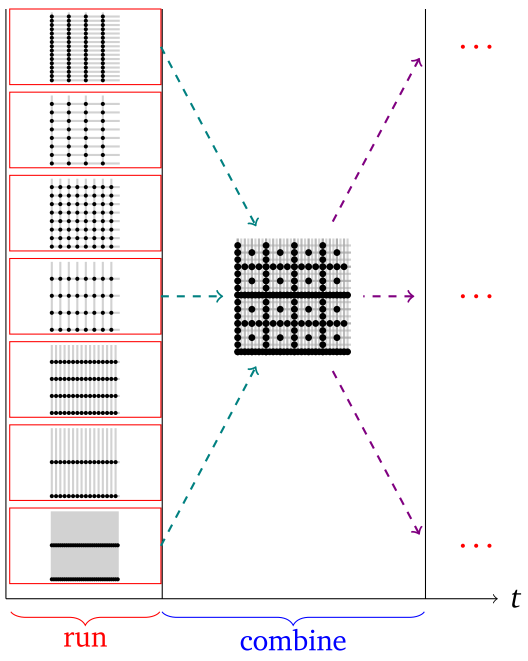


Figure 2.5.: Schematic process time of a CT time-stepping scheme: The solver update as well as the basis transforms are embarrassingly parallel between the component grids. Synchronization happens for the combination step, when data is gathered from the component grids and scattered back for the next solver update.

Figure 2.5 subsumes the recombination as one set of arrows, and the decombination as another set of arrows. It also illustrates the advantage of the embarrassing parallelism between the solver steps on the component grids: Solver time steps are entirely data-independent between component grids. The DisCoTec code used for this work, cf. Chapter 4, is designed to perform exactly these types of time stepping simulations.

An in-depth analysis on various error terms introduced by the time-dependent CT for finite-difference advection solvers can be found in [104, 144]. In particular, the leading error term that is introduced by the CT can be dampened out by choosing a sufficiently small combination interval length (which is validated in Section 3.4.4). However, this damping effect only works if the advection direction only changes very little within the combination interval [104].

Asynchronous Combination Technique

The asynchronous CT [119, section 3.5] is a promising extension, which overlaps computation of the solver update with communication of the combined solution.

Instead of waiting for the combination operation after time step T to finish, the combination operation is started on a copy of the current data, and the actual data structure containing an inexact solution $\tilde{u}_{\vec{\ell}}^{T}$ (without correction by the other component grids) can already be used for the next solver time step. The (inexact) solution of this time step is denoted $v^{\tilde{T}+1}$. Only after the time step has finished, the groups use the received combined data $u_{\vec{\ell}}{}^{T}$ to correct the data for the next time step:

$$u_{\vec{\ell}}{}^{T+1} \approx \tilde{v}^{T+1} + \Delta \,\text{with}\, \Delta = u_{\vec{\ell}}{}^{T} - \tilde{u}_{\vec{\ell}}^{T+1}. \tag{2.30}$$

Since three copies of the full grid data $u_{\vec{\ell}}$ are required, this method has a considerable total memory overhead. At the same time, the data copies allow 'rolling back' the solver time step update in case the correction term Δ is too large, which adds resilience to the scheme. The asynchronous CT was not used for this thesis, but could provide a particularly interesting extension in the context of the widely-distributed CT, cf. Section 4.5, which is characterized by very long communication latencies.

Dimensionally-Adaptive and Spatially-Adaptive Combination Technique

With the dimensionally-adaptive CT, the index set $\mathcal{I}^{\text{CT}}$ is specifically constructed for the function f that should be integrated/interpolated/represented with the CT. In this context, 'adaptivity' can mean automatic adaptation of $\mathcal{I}^{\text{SG}}$ by way of different error indicators [53], but also entails manual selection of the index set according to domain knowledge—for instance to make sure that the CT contains resolution ranges of interest for the Vlasov code at hand. This way of adapting in the CT means that one is still working on regular structured grids, with combination coefficients determined by Equation (2.27).

Although it was not used for the simulations presented in this thesis, the spatially-adaptive CT [121] is another—very promising—extension. It allows combining component grids with octree-like refinement structures in the subdomains of interest, per component grid. As a result, the combination is performed on block-structured grids of different base resolutions, and the refinement is chosen such that the combination coefficients $c_{\vec{\ell}}^{c}$ per component grid need not be changed.

2.4. Summary

This chapter introduced the mathematical background of the CT. It allows solvers to be oblivious to any multi-scale operations, providing a separation of concerns between solver and data exchange between the grids Two features of the CT will be pivotal to the rest of the thesis: Firstly, the intermediate hierarchical representation allows for some flexibility, and the already mentioned benefits of the mass-conserving variant will be explored in Chapter 3. Secondly, the embarrassing parallelism in the solver update allows for an added level of parallelism, which will be analyzed in Chapter 4 and experimentally validated in Chapter 5.

Chapter 3

Multiscale Bases for Accuracy, Conservation, and Numerical Stability

The hierarchical hat basis functions described in Section 2.1.1 can be transformed to the nodal basis at low computational and communication cost, due to the interpolating property ('interpolets') [97]. However, there can be problems when using them for time-stepping computations: Even though the recombination—the transformation from the individual component grids to a common sparse grid representation—is mass-conserving, the same does not necessarily hold for the decombination step. This chapter shows—by three different setups—that a choice of different hierarchical bases can provide conservation and accuracy in combination technique (CT) partial differential equation (PDE) solutions, and even enables simulations that would become unstable when using the hierarchical hat basis.

This chapter starts with a discussion of the conservation of mass in the context of sparse grids and the CT as well as other multiscale methods. It goes on to define terminology and error measures, and illustrates the mass conservation and stability properties by two-dimensional examples. The accuracy of the different variants of basis functions is evaluated with a two- to six-dimensional advection scenario. Improved numerical stability can be observed for the Vlasov–Poisson equations in two different six-dimensional benchmark scenarios. The chapter concludes with a practical assessment of the new basis functions.

The simulations in Sections 3.4 and 3.5 were conceptualized and performed by the author of this work and first published in [137]. For Section 3.5, Katharina Kormann contributed the modeling and code of SeLaLib as well as the concrete scenarios and validation to [137]. The results presented here are a summary of the aforementioned publication, with additional context and discussion.

3.1. Related Work: Multiscale Methods, Conservation, and Sparse Grids

The conservation of mass and higher-order moments is a critical question for higher-dimensional high-fidelity numerical solvers. In the context of sparse grids, the typical approach is to account for conservation in the hierarchical sparse grid solver, like in [60, 64, 97, 157, 166]. Koster, whose work is cited extensively in Chapter 2, uses the biorthogonal basis function to simulate turbulent flow problems on a spatially adaptive sparse grid [60, 97]. Kormann and Sonnendrücker [95] used ψ^{hat} basis operators in the SeLaLib code, while employing higher-order polynomials for the Vlasov–Poisson equations with a multiplicative δf ansatz. The work did not (yet) consider conservation in the sparse grid solution. This holds also true for the work by Deriaz and Peirani [32], who used hierarchical hat functions as a basis for a finite-difference Vlasov–Poisson sparse grid solver. Of the aforementioned, Guo and Cheng in particular use adaptive Alpert multiwavelets [3] to achieve conservation of mass and higher-order moments for Vlasov solvers, at the cost of dropping the continuity requirement on the solution—which also enables them to use discontinuous Galerkin methods. Using multiwavelets as basis functions appears promising for the CT as well [96].

Previous CT plasma simulations were able to successfully extrapolate eigenvalues for the linear part of the simulation [100]. The CT also worked nicely for gyrokinetic GENE simulations with some adaptation in the (de)hierarchization operators, when simulating the linear growth phase of the instability scenario [101]. However, nonlinear simulations with GENE would often become numerically unstable when using the CT [102, p. 319].

To the author's best knowledge, the only works that address conservation in the context of time-stepping sparse grid combination technique simulations are by Huber [83] and Zeiser [172]. Huber [83] custom-tailors geometry- and problem-dependent stencil operators in a staggered-grid fluid solver to conserve the solution's divergence to first order and to stabilize the simulation. Zeiser [172], on the other hand, stabilizes a discontinuous-Galerkin simulation with streamline diffusion for the transport equation. The paper further describes the application of a GPU-accelerated FEM library for relatively flexible geometries, with implicit higher-order time evolution in the solver. By this approach, different invariants of a Vlasov–Poisson scenario (mass, total energy, entropy) can be conserved.

Are Adaptive Biorthogonal Wavelets and Adaptive Sparse Grids the Same?

There is an enormous amount of literature on adaptive and/or conserving wavelet solvers that is potentially relevant to this chapter once the equivalence of hierarchical basis functions and lifting wavelets is established. This paragraph focuses on approaches that either address high-fidelity plasma applications or the intersection of conservation and adaptivity in wavelet methods, or all of these factors.

A group around Alam, Kevlahan, and Vasilyev extensively researched adaptive collocation methods with biorthogonal wavelets for different application PDEs [1, 38, 141]. Recent work on climate modelling stresses both the importance and the tractability of solving conservative large-scale two- and three-dimensional problems with adaptive biorthogonal wavelets [91].

Strains of work by Haefele, Latu, and Gutnic [65, 66, 68] and Besse et al. [12] focus on adaptive wavelet methods for the Vlasov–Poisson system. They are particularly interesting in comparison to the results in Section 3.5 since they too use a semi-Lagrangian scheme as a solver. Both lines of work started with a construction based on interpolets (hat functions $\psi^{\mathtt{hat}}$), but only one of them considered $\psi^{\mathtt{bo}}$ in a later paper [65]. Also, joint work by Besse, Deriaz, and Madaule considers an adaptive Alpert multiwavelet solver for the solution of the Vlasov equations [11], similarly to Guo and Cheng [64].

In the works listed here, all solvers operate in the hierarchical basis, but none explicitly uses a sparse grid construction—and the exact nature of the adaptive refinement is seldom discussed. So there remains a question if only uniform adaptive refinement (per cell) is allowed, or if directional refinement ('semicoarsening') is permitted as well. If the latter was the case, then the resulting data structures should be very similar to spatially adaptive sparse grids [125]. So another way to pose this question is:

> Does it have *octrees* or does it have *sparse grids*?

Potentially, both would be possible from most of the textual descriptions. But the overwhelming presence of octree-type refinement figures rather suggests that *if a paper does not mention sparse grids, it uses octrees*.

If this proposition is true, this could be unfortunate, since the main complexity of hierarchical sparse grid PDE methods lies in the adaptive hierarchical operators, which these authors have already successfully implemented and tested. At the same time, much could be gained from the missing part—by allowing semicoarsened (sparse grid) spaces in the refinement process—since the sparse grid benefits discussed in Section 2.2 still hold when only considering a small sub-cell of the domain. This would result in similar methodologies to the one pursued by Guo and Cheng [64].

So for these existing solvers that are likely based on octrees, an easier adaptable alternative would be to use them in conjunction with the spatially adaptive CT [119], which operates on block-structured octree-like component grids. The adaptive CT enables using the existing adaptive solvers with minimal changes, while still benefitting from the sparse grid structure. And since the solvers already operate on hierarchical bases, no intermediate transformations would be necessary for recombination and decombination.

3.2. Prelude: Quantities of Interest and Error Measures

When dealing with higher-dimensional simulations, it is inherently hard to visualize the solution. Thus, in order to draw meaning from these simulations, one often uses lower-dimensional quantities of interest Q that are derived from the solution. For example in Vlasov equations, these quantities often include the potential energy or the radial heat flux. This chapter typically uses the combined quantities of interest

$$Q_{\mathrm{ct}}(f) := \sum_{\vec{\ell} \in \mathcal{I}^{\mathrm{CT}}} c^c_{\vec{\ell}} \cdot Q(f_{\vec{\ell}}), \tag{3.1}$$

which are only the same as the sparse grid quantities of interest

$$Q_{\mathrm{sg}}(f) := Q\left(\sum_{\vec{\ell} \in \mathcal{I}^{\mathrm{CT}}} c^c_{\vec{\ell}} \cdot f_{\vec{\ell}} \right) \tag{3.2}$$

for linear operators such as the mass

$$m(f) := \|f\|_1 = \int_\Omega f(\vec{x})\mathrm{d}\vec{x}. \tag{3.3}$$

This last definition assumes the positivity of f, which is typically given for the simulations considered here (apart from the times when numerical instabilities arise).

For SeLaLib's potential energy—a non-linear quantity—Equation (3.1) and Equation (3.2) are not the same. Yet, it could be validated that they closely match in the numerically stable regions of the simulation by small combination scheme simulations, where one can interpolate the solution onto a 'fine' grid of level $\vec{\ell}^{\max}$ and use SeLaLib to compute the energy there.

If an exact solution f_{exact} is known, it is possible to measure the function error norms $\|f_{\text{exact}} - f\|$. In our setting, it is important to not only consider the sparse grid points, but to also consider the error 'in between' them. After all, there is a lot of space between sparse grid points in higher dimensions for high function errors to occur. The approach of this chapter is to use relative Monte Carlo error integrals

$$\frac{\|f^{\text{CT}} - f_{\text{exact}}\|_2}{\|f_{\text{exact}}\|_2}(t) \approx \frac{\sum_{j=1}^{M} \left|f^{\text{CT}}(\vec{x}_j,t) - f_{\text{exact}}(\vec{x}_j,t)\right|^2}{\sum_{j=1}^{M} \left|f_{\text{exact}}(\vec{x}_j,t)\right|^2}, \tag{3.4}$$

to get a (relatively dimension-independent) measure of the error. Through the combination formula (2.27), this error can be directly related to multilevel Monte Carlo error measures. The number of samples M was set to 10^5 and the pseudorandom coordinates $\vec{x}_j$ were obtained with the Mersenne Prime Twister algorithm. Since the interpolation of $f^{\text{CT}}(\vec{x}_j, t)$ is a linear operation, the CT Monte Carlo errors in Equation (3.4) and their sparse grid equivalents are the same.

3.3. 2D Examples: Tricks With Hierarchical Hat Basis Functions

Suppose that a mass-conserving PDE solver is used with the CT. It has periodic boundary conditions on the two-dimensional domain $[0,1)^2$, and the initial solution is constant $f_0(x_1,x_2) = 1$, so the integral / L_1 norm / mass of f_0 is 1, compare Equation (3.3). This function can be represented on any linear component grid. The following—very simple—two-dimensional combination scheme $\mathcal{I}^{\text{CT}}$ is used:

Combination scheme for 2D examples

$\vec{\ell}^{\min}$	$(1,1)$	# grids	5
$\vec{\ell}^{\max}$	$(3,3)$	# finest grids	3
total FG #DOF	6.40×10^1	mem. finest grids	128 B
total FG memory	512 B	#DOF FG at $\vec{\ell}^{\max}$	6.40×10^1

A few words about this representation of combination schemes, since it will be used throughout the thesis:

- The two-dimensional scheme contains five component grids, three of which are on the main diagonal and thus have the finest resolution, cf. Figure 2.4a.
- The grids on the main diagonal have a memory footprint of 16 DOF or 128 B. The other grids have half of these figures.
- This means that the full grids in the scheme have a total of 64 DOF, which leads a total full grid memory of 512 B, since each double-precision DOF has eight bytes.
- If we were to use only the full grid at resolution level $\vec{\ell}^{\max}$, it would also have 64 DOF. This shows us that the CT is not necessarily more efficient than using the full grid. In this example the memory footprint is the same, but the CT's memory requirement would already be smaller for a slightly higher difference between $\vec{\ell}^{\min}$ and $\vec{\ell}^{\max}$, or for more than three dimensions.

Now we look at two carefully constructed examples of possible analytical solutions at a later time t and how the solver tries to represent them on each component grid. Each illustrates a potential problem for a PDE solver that arises from using hierarchical hat functions, which can be solved by using the mass-conserving variants introduced in Section 2.1.1.

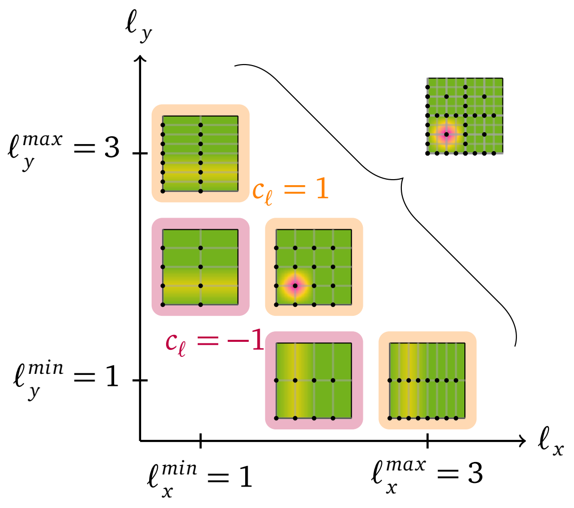
3.3.1. The Vanish Trick, or Making All Mass Disappear

Say that after one time step, the exact solution would be the piecewise bi-linear function

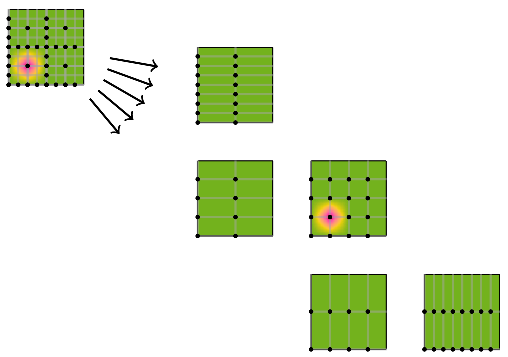
$$f_1(x_1, x_2) = 16 \cdot \phi^{\mathtt{hat}}_{1/2}(x_1) \cdot \phi^{\mathtt{hat}}_{1/2}(x_2); \tag{3.5}$$

recall the notation for nodal basis functions in Equation (2.2). The function has an integral of 1—so a mass-conserving solver should conserve a mass of 1.

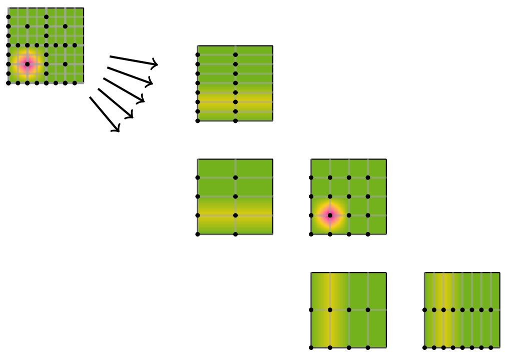
We employ a CT with $\vec{\ell}^{\min} = (1,1)$ and $\vec{\ell}^{\max} = (3,3)$. This means that out of the five full grids used, only one can exactly represent the solution, see the grid with $\vec{\ell} = (2,2)$ in Figure 3.1.



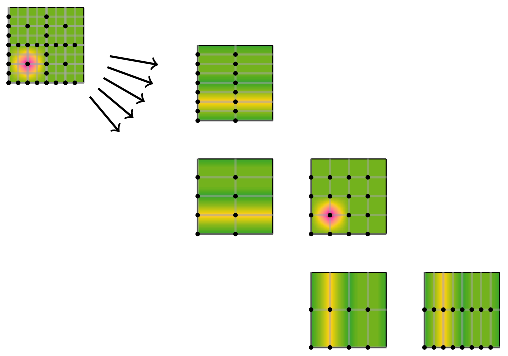
(a) Recombination step



(b) Decombination step with $\psi^{\mathtt{hat}}$



(c) Decombination step with $\psi^{\mathtt{bo}}$



(d) Decombination step with $\psi^{\mathtt{fw}}$

Figure 3.1.: Two-dimensional example of recombination with $\vec{\ell}^{\min} = (1,1)$ and $\vec{\ell}^{\max} = (3,3)$. The exact solution is given by Equation (3.5).

The other four grids just do not contain the solution's maximum point at $(0.25, 0.25)$. Instead, we assume for those grids that the solver does the next best thing—it places the available mass as close as possible to the true maximum point. Then, due to the low resolution and periodic boundaries, the solution on those grids will be constant in one dimension, either $f(x_1,x_2)_1 = 4 \cdot \phi^{\mathtt{hat}}_{1/2}(x_1)$, or $f(x_1,x_2)_1 = 4 \cdot \phi^{\mathtt{hat}}_{1/2}(x_2)$. Now, the combined solution is exactly equal to the true solution, since the 'under-resolution' errors are cancelled pairwise (see the left and lower grids in Figure 3.1a respectively). This is an example of the CT's error cancellation, cf. Section 2.3.1. And indeed, the combined sparse grid solution is the 'true' solution. Note that the recombination step is always mass-conserving for a mass-conserving solver due to the linearity of the mass. But a problem arises when we use the *injection* operation associated with ψ^{hat}, cf. Section 2.1.1, to populate the component grids again (decombination): Since only one sparse grid point contributes mass, cf. Figure 3.1b, the four grids that do not contain it will be empty. Notably, the combined sparse grid still holds the true solution. But for the next time step, four out of five grids will not contribute to the sparse grid solution anymore, it will effectively be a computation on the grid $(2,2)$ only.

Thus, the example illustrates that the decombination step is not mass-conserving for the component grids when using the hierarchical hat basis, and this is clearly something to avoid for mass-conserving PDE solvers. The biorthogonal basis function and full weighting basis function, by comparison, accurately conserve the mass on the component grids.

3.3.2. The Explosion Trick, or Amplifying Mass and Gradients

For the second example, one needs to consider two combination time steps. Again, the solver tries to represent the exact solution, which is now given by

$$f_2(x_1, x_2, t_1) = 4 \cdot (\phi^{\text{hat}}_{0/3}(x_2) + \phi^{\text{hat}}_{4/3}(x_2)) \tag{3.6}$$

after the first time step.

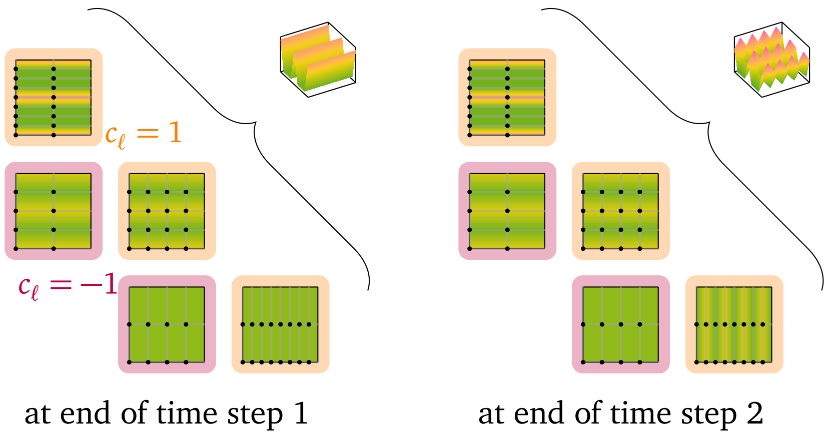


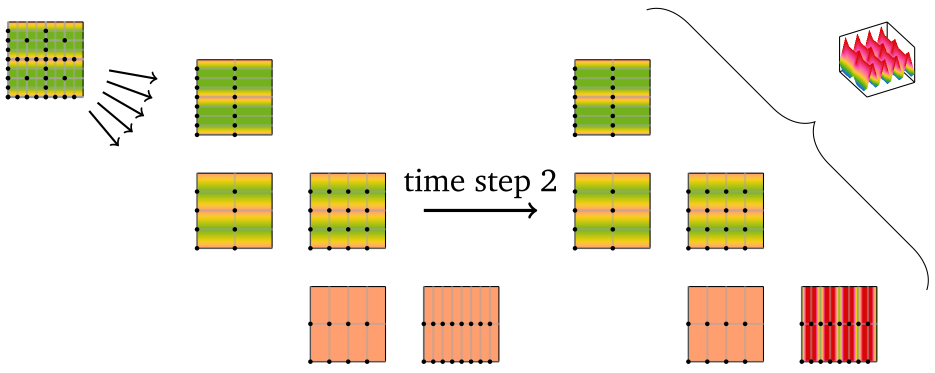
Figure 3.2.: The ideal 'true' solutions after the first and second time step

The second time step introduces a high-frequent (mass-conserving) mode in the transversal x_1 direction, such that the updated solution is

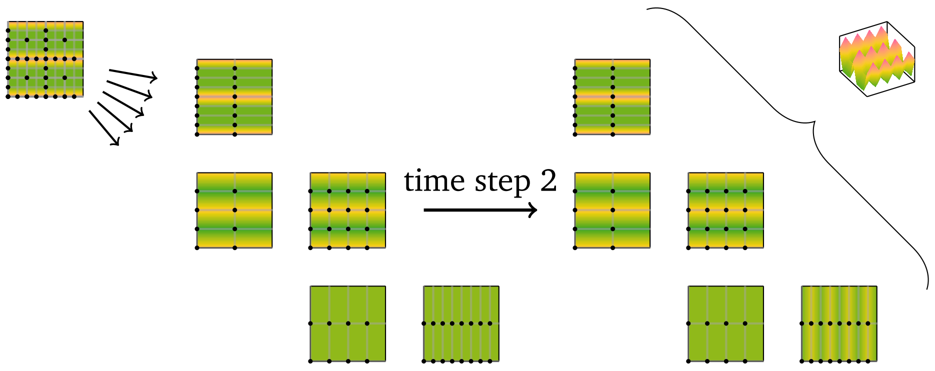
$$\begin{aligned} f(x_1, x_2, t_2) = 4 \cdot (\phi^{\mathtt{hat}}_{0/3}(x_2) + \phi^{\mathtt{hat}}_{4/3}(x_2)) & \qquad (3.7)\\ + 1 \cdot (\phi^{\mathtt{hat}}_{1/3}(x_1) + \phi^{\mathtt{hat}}_{3/3}(x_1) + \phi^{\mathtt{hat}}_{5/3}(x_1) + \phi^{\mathtt{hat}}_{7/3}(x_1)) & \qquad (3.8)\\ - 1 \cdot (\phi^{\mathtt{hat}}_{0/3}(x_1) + \phi^{\mathtt{hat}}_{2/3}(x_1) + \phi^{\mathtt{hat}}_{4/3}(x_1) + \phi^{\mathtt{hat}}_{6/3}(x_1)), & \qquad (3.9) \end{aligned}$$

which most grids cannot resolve, and thus they stay unchanged to conserve the mass. Only the grid $(3, 1)$ can resolve the new mode, as can be seen in the right of Figure 3.2, which allows the mode to occur in the combined sparse grid. This would be the ideal case, in which the true solution can be replicated.

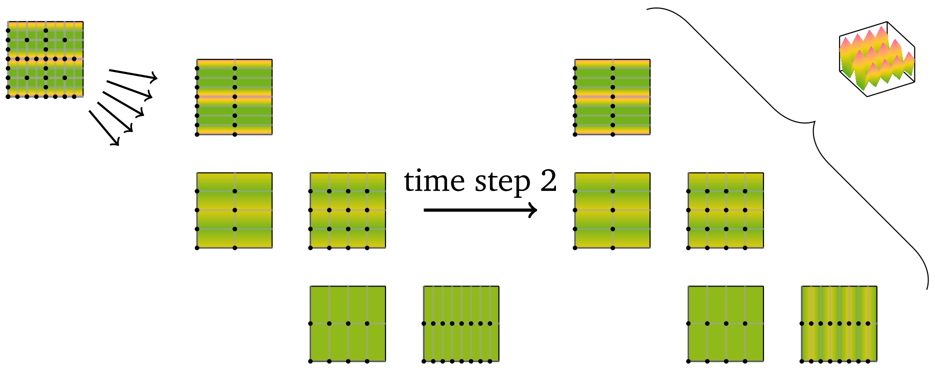
Figure 3.3a shows what happens for $\psi^{\mathtt{hat}}$: Due to the lack in mass conservation, the decombination after the first time step amplifies the mass in some component grids, to obtain up to four times the mass (cf. the constant orange functions). In fact, for arbitrary level differences $\vec{\ell}^{\max} - \vec{\ell}^{\min}$, the mass in the component grids can be multiplied by up to a factor of $2^{\ell^{\max}_j - \ell^{\min}_j}$ in dimension j. As the second time step introduces its high-frequent mode on just one of these amplified grids, very high and low absolute values ensue in the combined solution, along with gradients that are up to $2^{\vec{\ell}^{\max}{}_2 - \vec{\ell}^{\min}{}_2}$ times higher than in the original solution Equation (3.7). Figures 3.3b and 3.3c shows that $\psi^{\mathtt{bo}}$ and $\psi^{\mathtt{fw}}$ do not suffer from this effect, because the mass is conserved in the decombination step. The combined solution for the mass-conserving basis functions is the exact solution (3.7), and for $\psi^{\mathtt{fw}}$, even the component grid solutions look exactly like the 'desired' solutions. There is a symmetry between this example and the previous one: Here, $\psi^{\mathtt{fw}}$ can capture the desired solution exactly and conserves positivity on the component grids, while $\psi^{\mathtt{bo}}$ leads to some negative function values on the component grids—and vice versa for Section 3.3.1. Note that the overall mass, being a linear quantity of interest, is still conserved in every combined solution. For $\psi^{\mathtt{hat}}$ it diverges on the component grids only—in a way, the error cancellation (Section 2.3.1) still applies to the combined solution. But this example shows that, for arbitrarily high level differences, unbounded mass and gradients can occur if $\psi^{\mathtt{hat}}$ is used for recombination; In fact, the instability of the basis and the lack of conservation of mass can be considered two sides of the same coin, as they are both a consequence of the lack of regularity in the hierarchical hat's dual 'function', the delta distribution (cf. Section 2.1.2).



(a) second time step with $\psi^{\texttt{hat}}$

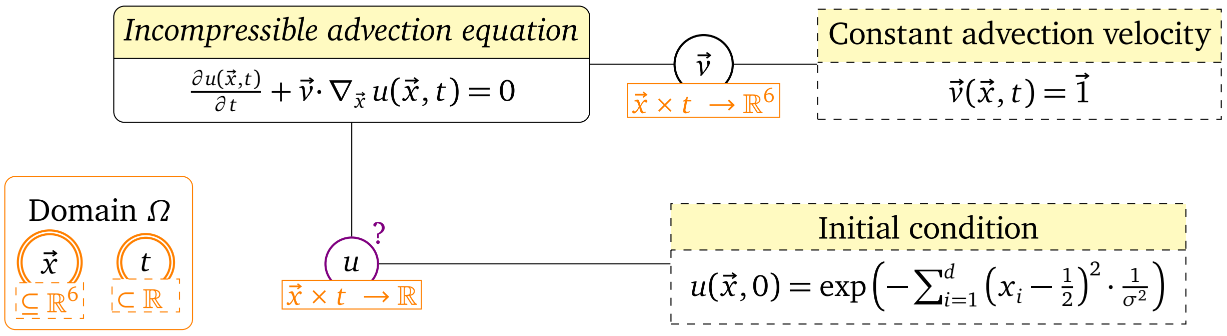


(b) second time step with $\psi^{\texttt{bo}}$



(c) second time step with $\psi^{\texttt{fw}}$

Figure 3.3.: Two-dimensional example of recombination with $\vec{\ell}^{\min} = (1,1)$ and $\vec{\ell}^{\max} = (3,3)$. The exact solutions are given by Equation (3.6).



Incompressible advection equation
$\frac{\partial u(\vec{x},t)}{\partial t} + \vec{v} \cdot \nabla_{\vec{x}} u(\vec{x},t) = 0$
$\vec{v}$
$\vec{x} \times t \to \mathbb{R}^6$
Constant advection velocity
$\vec{v}(\vec{x},t) = \vec{1}$
Domain Ω
$\vec{x}$
$\subseteq \mathbb{R}^6$
t
$\subset \mathbb{R}$
u
?
$\vec{x} \times t \to \mathbb{R}$
Initial condition
$u(\vec{x},0) = \exp\left(-\sum_{i=1}^{d}\left(x_i - \frac{1}{2}\right)^2 \cdot \frac{1}{\sigma^2}\right)$

Figure 3.4.: Extended Model Pathway Diagram [93] of the constant incompressible advection equation: Quantities are given in circles. The orange boxes denote the types of the quantities. Double circles denote the variables of the computational domain. Dashed black lines indicate constants and boundary conditions. Purple circles with question marks denote the unknown quantities that need to be solved for. In comparison to Figure 1.1, it becomes apparent that the model is simpler to solve than the Vlasov–Poisson system, as there is no cycle in the advection diagram.

3.4. Conservation of Mass and Increased Accuracy for Advection in $2-6D$

Let us consider the incompressible advection problem

$$\frac{\partial u(\vec{x},t)}{\partial t} + \vec{v} \cdot \nabla_{\vec{x}} u(\vec{x},t) = 0 \tag{3.10}$$

on the d-dimensional unit hypercube $\Omega = [0,1]^d$. Some initial concentration $u(\vec{x},0) = u_0(\vec{x})$ is advected with a constant velocity $\vec{v} \in \mathbb{R}^d$. Assuming periodic boundary conditions, the analytical solution can be obtained by translating the initial profile u_0.

The model pathway diagram of the incompressible advection model is shown in Figure 3.4. By comparing to Figure 1.1, one can already see that an advection problem in two to six dimensions is significantly simpler to model (and solve) than the Vlasov–Poisson system. Yet, it can serve as a good proxy for higher-dimensional Vlasov simulations, since both are kinetic transport models. Furthermore, periodic boundary conditions are used for grid-based Vlasov solvers like GENE [99] and SeLaLib [94]. And choosing a Gaussian function as initial condition

$$u_0(\vec{x}) = \exp\left(-\sum_{i=1}^{d}\left(x_i - \frac{1}{2}\right)^2 \cdot \frac{1}{\sigma^2}\right), \tag{3.11}$$

is also close to Vlasov benchmark problems, where sums of Gaussians are the typical initial conditions, and also the typical steady state solutions. Gaussian functions have proven to be particularly challenging to represent on sparse grids with piecewise linear basis functions [125].

The advection problem has an analytical solution, such that the numerical errors can be evaluated—in contrast to turbulent Vlasov simulations, where the exact analytical solution is typically unknown.

3.4.1. Finite Volume / Finite Difference Discretization

For simplicity, we use a constant uniform velocity $\vec{v} = (1, 1, \ldots, 1)$. This allows us to discretize the advection equation with first order accuracy by

$$\left.\begin{aligned} \nabla_i u = \frac{\partial u}{\partial x_i} &\approx \frac{u(\vec{x}, t - \Delta t) - u(\vec{x} - \Delta x_i, t - \Delta t)}{\Delta x_i} \\ \frac{\partial u}{\partial t} &\approx \frac{u(\vec{x}, t) - u(\vec{x}, t - \Delta t)}{\Delta t} \end{aligned}\right. \Rightarrow u(\vec{x}, t) = u(\vec{x}, t - \Delta t) - \nabla u \cdot \vec{v} \cdot \Delta t. \tag{3.12}$$

The discretization can be understood both as first-order finite difference scheme with backward differences, or as finite volume discretization with upwinding—both result in the same stencil values. The last line denotes an explicit Euler time stepping approach. The accuracy will depend linearly on the resolutions both in space $\vec{x}$ and time t. Finite volume upwinding conserves the mass m, cf. Equation (3.3), up to machine precision. The solver time step size Δt is selected as 1×10^{-4} in order fulfill the CFL condition for all resolutions considered; The CFL condition [28] relates the maximum time step length to the spatial resolution for explicit schemes—the finer the resolution, the shorter the time step needs to be. The numerical mass at $t = 0$ will vary slightly depending on the resolution, due to the numerical interpolation of the solution (3.11) during initialization.

3.4.2. Conservation of Mass on Component Grids

As illustrated in Section 3.3, the combined mass in the simulation will be conserved by the CT, even if the mass is not conserved on the individual component grids. The same is true for the advection problem: If we consider the three-dimensional combination scheme $\vec{\ell}^{\min} = (2, 2, 2)$, $\vec{\ell}^{\max} = (11, 11, 11)$ with recombination after every time step, then the combined mass will fluctuate only by about 2×10^{-14} around the initial value for all three hierarchical basis functions [137]. But the mass on the component grids behaves very differently.

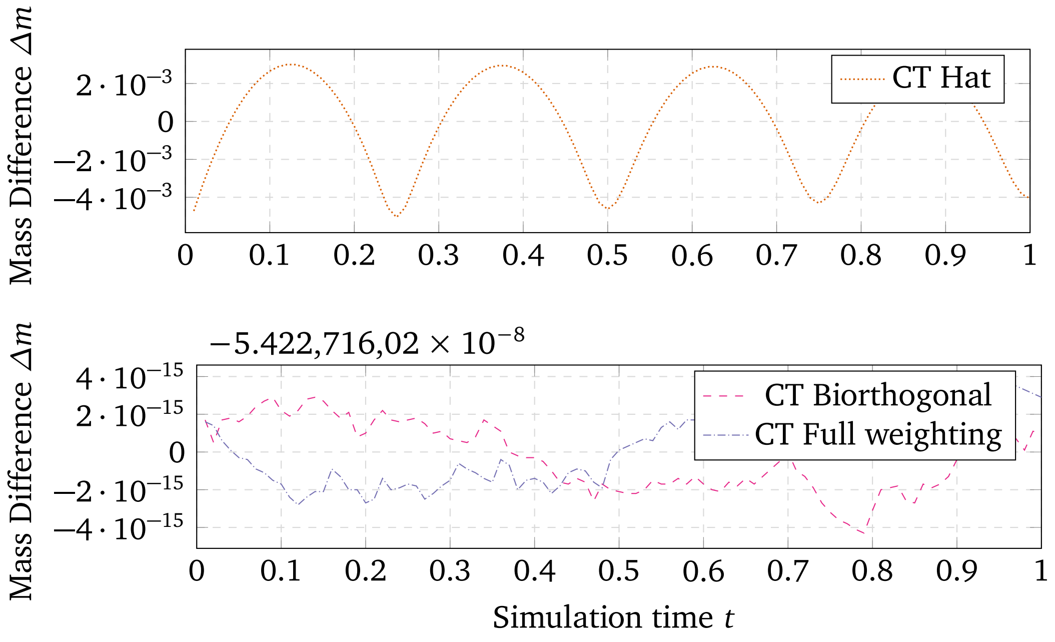


Figure 3.5.: Mass difference Δm between the simulated and analytical mass $\int u^{\text{exact}}(\vec{x}, \cdot)\, d\vec{x} \approx 0.186$ for a single component grid. The entire CT advection scenario with $\vec{\ell}^{\min} = (2, 2, 2)$, $\vec{\ell}^{\max} = (11, 11, 11)$ contains 136 component grids. The graphs show the mass on the component grid with $\vec{\ell} = (9, 2, 2)$, one of the coarsest resolved in the scheme. Using hierarchical hat functions leads to fluctuations of up to 2.7% of the analytical mass. By comparison, the biorthogonal and full weighting bases conserve the mass in the grid up to an accuracy of 1×10^{-14}. (Figure first published in [137])

Figure 3.5 shows the mass if evaluated only on the component grid with $\vec{\ell} = (9, 2, 2)$. While the mass stays well within the expected machine precision for combination with $\psi^{\texttt{bo}}$ and $\psi^{\texttt{fw}}$, it changes by more than 10^3 with $\psi^{\texttt{hat}}$. This shows that the idealized result from Section 3.3 is transferrable to the advection problem and that mass-conserving hierarchical functions are indeed necessary to achieve mass conservation on the component grids.

3.4.3. Improved Accuracy with Mass-Conserving Functions

Furthermore, the mass-conserving functions also improve the numerical accuracy. Figure 3.6 compares the decay of the relative Monte Carlo L_2 error (3.4) for different dimensionalities d, resolutions, and hierarchical basis functions. While the full grid reference simulations are each run on a single grid of isotropic resolution $\vec{\ell}^{\max}$, all the CT simulations have a minimum level $\vec{\ell}^{\min}$ of $(2)^d$, resulting in an increasing number of component grids as $\vec{\ell}^{\max}$ increases.

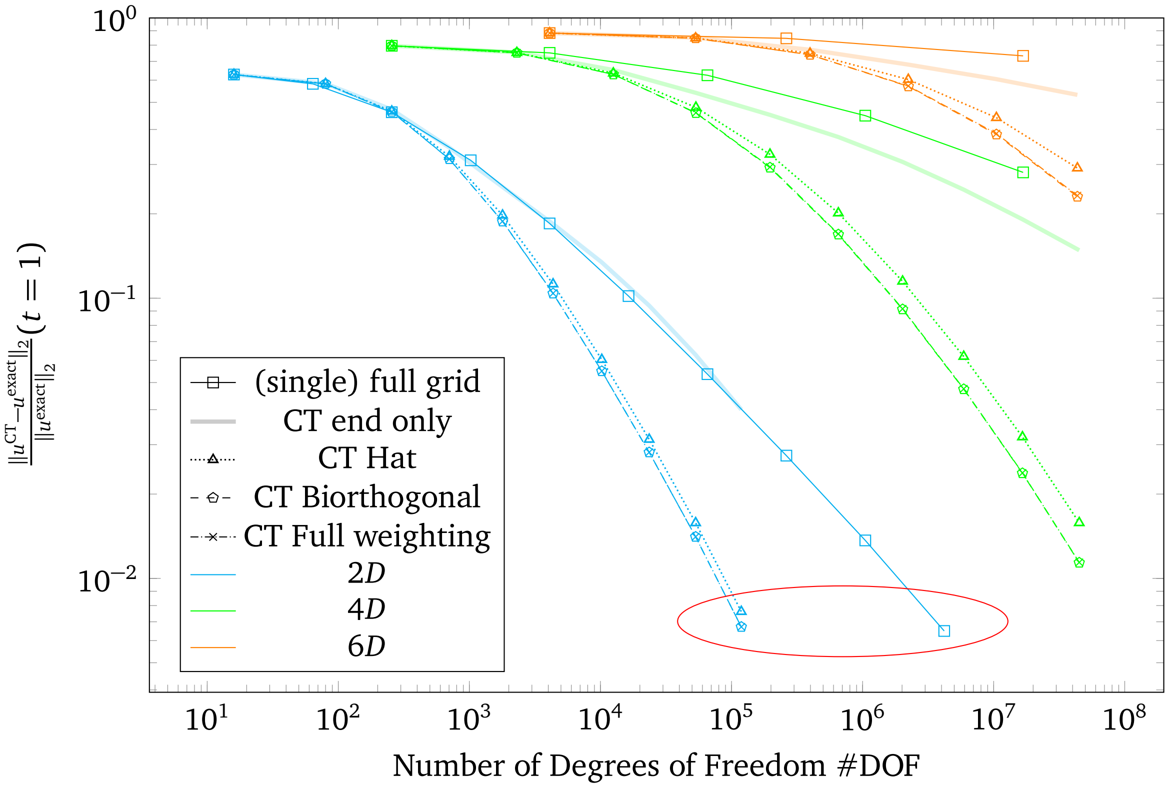


Figure 3.6.: Relative Monte Carlo error norms, cf. Equation (3.4), for the advected Gaussian profile over the number of DOF used. Results for full grid and different CT schemes are shown for dimensionalities of $d = 2, 4, 6$. The circled data points highlight the different simulations with the (maximum) resolution of $\vec{\ell}^{\max}$, which will be further compared in Section 3.4.4. (Figure first published in [137])

The 'largest' combination scenarios considered in two and six dimensions, respectively, are given by:

Combination scheme largest 2D advection setup

$\vec{\ell}^{\text{min}}$	$(2,2)$	# grids	19
$\vec{\ell}^{\text{max}}$	$(11,11)$	# finest grids	10
total FG #DOF	1.19×10^5	mem. finest grids	64 KiB
total FG memory	928 KiB	#DOF FG at $\vec{\ell}^{\text{max}}$	4.19×10^6

Combination scheme largest 6D advection setup

$\vec{\ell}^{\text{min}}$	$(2,2,2,2,2,2)$	# grids	462
$\vec{\ell}^{\text{max}}$	$(7,7,7,7,7,7)$	# finest grids	252
total FG #DOF	4.35×10^7	mem. finest grids	1 MiB
total FG memory	332 MiB	#DOF FG at $\vec{\ell}^{\text{max}}$	4.40×10^{12}

From the latter, one can see why the full grid simulations at $\vec{\ell}^{\text{max}}$ were not performed for all setups up to level $(11)^d$: for a simulation at $\vec{\ell} = (7,7,7,7,7,7)$, the full grid solution would require $\approx$ 32 TiB of memory, which was far beyond the capabilities of the single-node system used for the simulations.

Mainly, Figure 3.6 shows that the benefit through the CT (higher accuracy per DOF) is already substantial for the 2D case and even more so as one goes to higher dimensionalities. Furthermore, achieving the same (low) error as the full grid solution with level $\vec{\ell}^{\text{max}}$ is the best one could hope for in a CT simulation.

When comparing for a single maximum level $\vec{\ell}^{\text{max}}$—for instance, consider the 'largest 2D advection setup', which belongs to the set of the lowest blue points in Figure 3.6—we observe that the error gap between the combined and full solution can be reduced by a factor of ≈ 3 when using the mass-conserving functions in place of the hierarchical hat function. This effect is observable across dimensionalities and also holds for the Monte Carlo L_1 and $L_{\max}$ errors.

As predicted by theory, see Section 2.3, one can also observe that the higher regularity of ψ^{bo} and ψ^{hat} lands the mass-conserving CT's accuracy somewhere between the standard hat function CT and the ideal case of the full grid [92, 137].

3.4.4. Influence of Recombination Time Step Lengths

In the previous subsection, Figure 3.6 also showed the resulting errors if there are no intermediate recombinations (labelled 'CT end only'). If no intermediate recombinations are performed, then the asymptotic order of accuracy degrades approximately to the full grid solutions' order of accuracy. Naturally, this poses the question if there is any middle ground between combining only at the end and combining after every single solver time step.

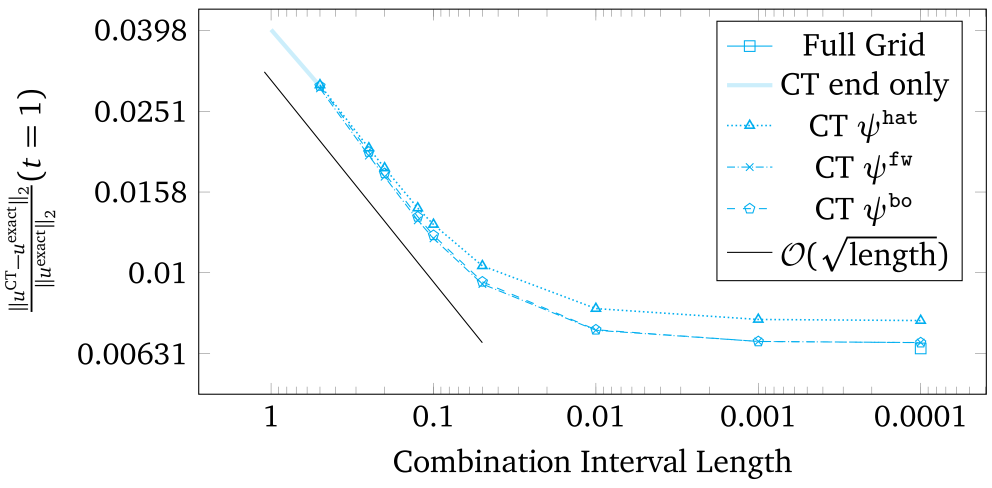


Figure 3.7.: Relative L_2 error for different recombination interval lengths to reach time $t = 1$. The time step in the advection solver is 1×10^{-4}. $\vec{\ell}^{\min}$ is $(2, 2)$ and $\vec{\ell}^{\max}$ is $(11, 11)$ for the CT solutions. The full grid solution is computed at $\vec{\ell}^{\max}$. The leftmost point here corresponds to the rightmost point of 'CT end only' / '2D' in Figure 3.6. The rightmost points here correspond to the circled data points in Figure 3.6. (Figure first published in [137])

Figure 3.7 investigates this question for the 'largest' $2d$ set-up with $\vec{\ell}^{\min} = (2, 2), \vec{\ell}^{\max} = (11, 11)$ (circled red in Figure 3.6). A significant error decrease can be observed starting from 1 (no intermediate combination) to 0.1. Then, the error approaches a constant level until at 1×10^{-4}, recombination happens after every solver step. This means that for this relatively smooth analytical solution with constant advection, one could choose the recombination interval considerably larger than the solver time step at a small increase in errors.

This—and also the square root behavior of the error—aligns with the numerical analysis by Lastdrager [104] as discussed in [137].

It is important to note that Lastdrager [104] observed this quick error decay only for regions of constant advection directions. Thus, the next sections cautiously recombine after every solver time step to accommodate for the potentially turbulent nature of Vlasov–Poisson solutions.

3.5. Stabilizing Plasma Simulations: Vlasov–Poisson with SeLaLib

This section introduces the basic Semi-Lagrangian idea to solving the Vlasov–Poisson equation on a grid—as implemented in the SeLaLib code, which proved to be well-suited for CT simulations. Then, the results are evaluated for CT simulations of two common plasma simulation benchmark problems: Landau damping and the two-stream instability, both with electrons in a neutralizing background. To provide a fair comparison between CT and full grid simulations, the setups are designed to occupy a similar amount of main memory for the distribution function f for both.

3.5.1. Semi-Lagrangian Method for Vlasov–Poisson Equations

The Vlasov–Poisson system of equations for a single particle species—in our case, electrons—reads

$$\partial_t f(\vec{x},\vec{v},t) + \vec{v}\cdot\nabla_{\vec{x}} f(\vec{x},\vec{v},t) + \frac{q}{m}\left(\vec{E}(\vec{x},t) + \vec{v}\times\vec{B}(\vec{x},t)\right)\cdot\nabla_{\vec{v}} f(\vec{x},\vec{v},t) = 0, \tag{3.13}$$

and

$$-\Delta_{\vec{x}}\phi(\vec{x},t) = 1 - \int f(\vec{x},\vec{v},t)\,\mathrm{d}\vec{v}, \quad \vec{E}(\vec{x},t) = -\nabla_{\vec{x}}\phi(\vec{x},t), \tag{3.14}$$

where q and m are the particle's charge and mass. The electric and magnetic fields are denoted by $\vec{E}$ and $\vec{B}$, respectively, and both $\vec{x}$ and $\vec{v}$ are three-dimensional. Poisson's equation describes the self-consistent fields for low-frequency phenomena. Recall from the corresponding model pathway diagram in Figure 1.1: Self-consistent means that there is a loop in the diagram and the equations depend on each other at all times t.

The general semi-Lagrangian idea denotes that one 'follows' the characteristic curves X, V along which the distribution function is constant:

$$f(\vec{x},\vec{v},t) = f_0(\vec{X}(0;\vec{x},\vec{v},t), \vec{V}(0;\vec{x},\vec{v},t)). \tag{3.15}$$

f_0 is the initial solution at time $\tau = 0$. The solution of the characteristic curve starting at $(\vec{x},\vec{v})$ at time τ is denoted by $(\vec{X}(\tau;\vec{x},\vec{v},t), \vec{V}(\tau;\vec{x},\vec{v},t))$ at some later time t. X and V can be computed with the characteristic ordinary differential equations

$$\frac{\mathrm{d}\vec{X}}{\mathrm{d}t} = \vec{V}, \quad \frac{\mathrm{d}\vec{V}}{\mathrm{d}t} = \frac{q}{m}\left(\vec{E} + \vec{V}\times\vec{B}\right). \tag{3.16}$$

Since the fields need to change self-consistently with the distribution function, the difference $t-\tau$ cannot be arbitrarily long for the same constant values of $\vec{B}$ and $\vec{E}$. Thus, there is an alternating iteration of solving the characteristic equations for the current coordinates $(\vec{x},\vec{v},t)$ and then 'looking into the past' what the (interpolated) value of f was at the location where the characteristic curve started at time τ. Here, we use Lagrangian interpolation due to its data locality, but other interpolation schemes are possible.

When using Cheng–Knorr splitting [21], the spatial and velocity advection steps alternate, such that the advection coefficients are constant within each update and the solution of the characteristic equations can be computed analytically.

The semi-Lagrangian method allows for time steps that are significantly larger than in Eulerian methods, and, by design, it conserves the probability 'mass' of f up to machine precision.

Details on the modeling, numerics, and scaling of the Vlasov–Poisson equation with SeLaLib can be found in [94]. SeLaLib's six-dimensional discretization is nested in a suitable way for the CT. This makes SeLaLib a good solver for application within the DisCoTec framework, which will be described in more detail in Chapter 4. For here, we content ourselves with a thorough assessment of the numerical simulation results, where a three-point Lagrange interpolation is used for the semi-Lagrangian method.

3.5.2. Landau Damping with DisCoTec + SeLaLib

Landau damping describes the exponential damping of longitudinal waves in plasmas [103]. To study it, the Vlasov–Poisson system was initialized with

$$f_0(\vec{x},\vec{v}) = \left(1+\varepsilon\sum_{i=1}^{3}\cos(kx_i)\right)\frac{1}{(2\pi)^{3/2}}\exp\left(-\frac{\|\vec{v}\|_2^2}{2}\right) \tag{3.17}$$

on the domain $[0,\frac{2\pi}{k}]^3\times\mathbb{R}^3$ for $k=0.5$.

The damping rate and the oscillation frequency in time can be approximated by linear dispersion analysis. For the mode $k=0.5$, the resulting damping rate is $\omega\approx-0.1533$. For small ε, e. g., $\varepsilon=0.01$, the prediction matches the observations well from the second oscillation onwards. Grid-based simulations incur the so-called numerical recurrence after a certain time depending on the resolution of the velocity grid [108]: A sudden growth in potential energy occurs at the recurrence time $T=\frac{2\pi}{k\Delta v}$ with Δv as the (minimal) resolution in velocity space.

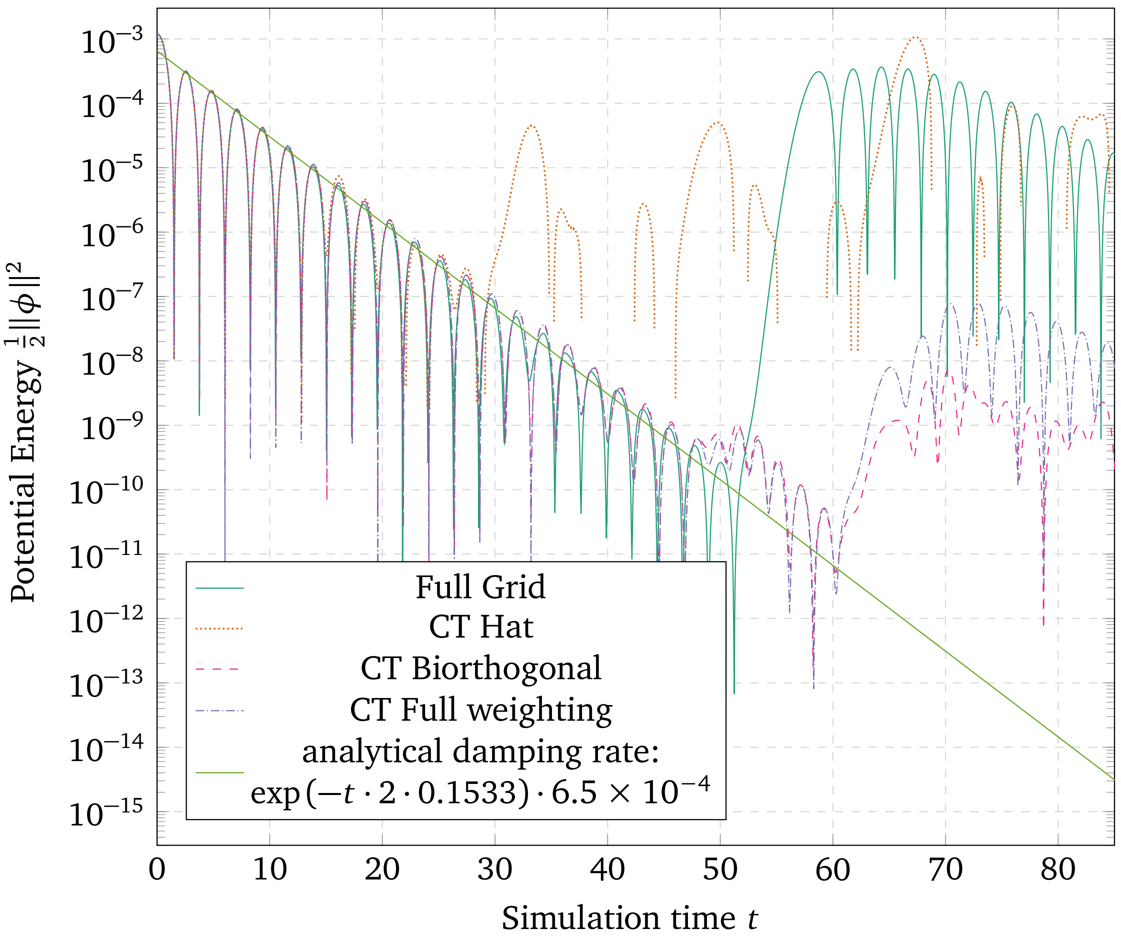


Figure 3.8.: Landau Damping: Comparison between CT and full grid solution, where the CT is allotted 5.4 GiB and the full grid 5.5 GiB for the distribution function data structures. In the CT simulations, combination takes place after every time step of $\Delta t = 0.01$. The numerical recurrence for the full grid solution occurs at $t \approx 59.7$, where it would be expected. (Figure first published in [137])

It was possible to validate the quick decay of the hierarchical coefficients α for finer-resolved spatial resolutions, such that the $\vec{x}$ levels are set to a relatively coarse uniform resolution of $\ell_{\vec{x}} = 4$, or 16 grid points, for this experiment.

Combination scheme for SeLaLib Landau damping

$\vec{\ell}^{\min}$	$(4,4,4,2,2,2)$	# grids	64
$\vec{\ell}^{\max}$	$(4,4,4,8,8,8)$	# finest grids	28
total FG #DOF	7.09×10^8	mem. finest grids	128 MiB
total FG memory	5 GiB	#DOF FG at $\vec{\ell}^{\max}$	6.87×10^{10}

Conclusively, the CT is effectively restricted to the three velocity components only, where each of the levels ranges from 2 to 8 (alternatively, 4 to 256 grid points), leading to 64 component grids in the CT. The pertaining data structures for the sixty-four different f would take up a total of 5.28 GiB. The CT is compared to a single full grid simulation, which shares the same spatial resolution and has 57 points in each of the velocity directions. Its distribution function data structure takes up 5.65 GiB, providing a relatively fair comparison.

Figure 3.8 shows some clear deviations from the analytical damping rate for all tested variants, which is likely due to the relatively low-order interpolation. One would expect the numerical recurrence in the full grid solution at around $\frac{2\pi\cdot 57}{0.5\cdot 12} \approx 59.7$, which can be observed in the figure. While no clear recurrence occurs for the CT solutions, smaller jumps can be observed. This happens already very early at around time 30 for the standard $\psi^{\mathtt{hat}}$, which also displays negative energy values and numerical instability quickly. However, the mass-conserving basis functions in the CT exhibit the jumps at a time similar to the recurrence on the full grid solution. One can conclude that the mass-conserving CT can capture the Landau damping phenomenon accurately: The numerical recurrence effect is not at all determined by the lowest velocity resolutions in the combination scheme, but the sparse grid points appear to be sufficiently well-placed to capture the phenomenon almost as well as the single full grid. Possibly, there could be an advantage for the CT if dimensional adaptivity was used to better resolve the most important velocity directions, cf. Section 2.3.2.

3.5.3. Two-Stream Instability with DisCoTec + SeLaLib

The two-stream plasma instability was excited in the DisCoTec + SeLaLib simulation by setting the initial distribution function to

$$\begin{aligned} f_0(\vec{x},\vec{v}) = &\left(1+\varepsilon\sum_{i=1}^{3}\cos(0.2x_i)\right)\frac{1}{2(2\pi)^{3/2}}\cdot \\ &\left(\exp\left(-\frac{(v_1-2.4)^2}{2}\right)+\exp\left(-\frac{(v_1+2.4)^2}{2}\right)\right)\exp\left(-\frac{v_2^2+v_3^2}{2}\right) \end{aligned} \tag{3.18}$$

on $\Omega = \left[0, \frac{2\pi}{0.2}\right]^3 \times \mathbb{R}^3$. The perturbation is set to $\varepsilon = 0.001$, and the potential energy is expected to show an exponential increase with a given growth rate of 0.2258 (the 'linear phase' of the simulation) after some initial oscillations. After some simulation time, nonlinear effects are bound to dominate and the electric energy will start to oscillate around a certain energy level, which denotes the nonlinear phase of the simulation. The oscillations are an effect of plasma particles getting trapped in the fields $\vec{B}$ and $\vec{E}$. The nonlinear phase of the instability simulation ideally displays turbulent but quasi-stationary behavior and is of particular interest for simulating controlled fusion plasmas, also for other types of plasma instabilities.

The initial condition (3.18) causes the two-stream instability to occur along the first dimension while the other dimensions should be characterized by Landau damping. When simulating the nonlinear phase, some commonly encountered numerical errors would be too high oscillations or damping of the electric energy, which was in fact a challenge in previous sparse grid approaches [95].

All simulations considered were run from $t = 0$ to $t = 200$ with a time step of $\Delta t = 0.01$. We consider a six-dimensional combination scheme with $\vec{\ell}^{\text{min}} = (3, 3, 3, 3, 3, 3)$ and $\vec{\ell}^{\text{max}} = (6, 6, 6, 6, 6, 6)$ (84 component grids).

Combination scheme for SeLaLib two-stream instability

$\vec{\ell}^{\text{min}}$	$(3,3,3,3,3,3)$	# grids	84
$\vec{\ell}^{\text{max}}$	$(6,6,6,6,6,6)$	# finest grids	56
total FG #DOF	1.43×10^{8}	mem. finest grids	16 MiB
total FG memory	1 GiB	#DOF FG at $\vec{\ell}^{\text{max}}$	6.87×10^{10}

It has approximately 1.1 GiB for the distribution function—the same amount of memory is required by the full grid solution with $\vec{N} = (22, 22, 22, 24, 24, 24)$. Additionally, we compare to a (rather expensive) finely-resolved full grid reference simulation of level $\vec{\ell}^{\text{max}}$, which takes up 512 GiB in memory for the plain f data. Since no analytical solution can be given for the turbulent nonlinear phase, this reference solution is considered as close to the 'truth'. The CT simulations perform the combination after each solver time step.

The resulting (combined) energy curves are displayed in Figure 3.9. While all approaches capture the linear phase as expected, the ψ^{hat} CT simulation displays deviating peaks early on and aborts at around $t = 106$ due to numerical instability. This does not at all happen for the other simulations considered, which oscillate around the reference energy level up to the end of the simulation. The finely-resolved reference solution needs 512 GiB and shows relatively low oscillations in the electric energy. The low amplitude of oscillations is slightly better matched by the mass-conserving CT solutions than by the full grid solution (green curve).

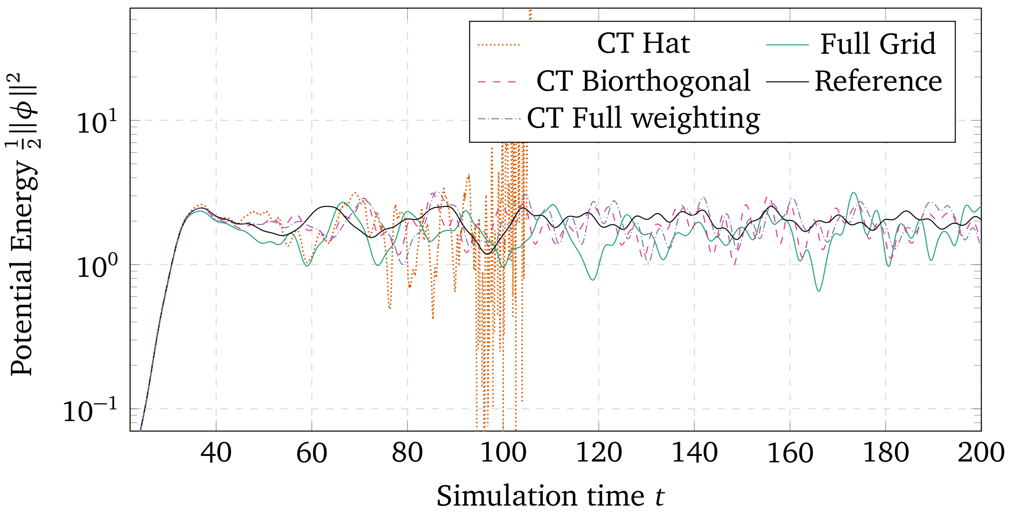


Figure 3.9.: The electric energy in the (combined) two-stream instability simulations. The CT simulations and the (green) full grid solution use up $\approx 1.1\,\text{GiB}$ for the distribution function.

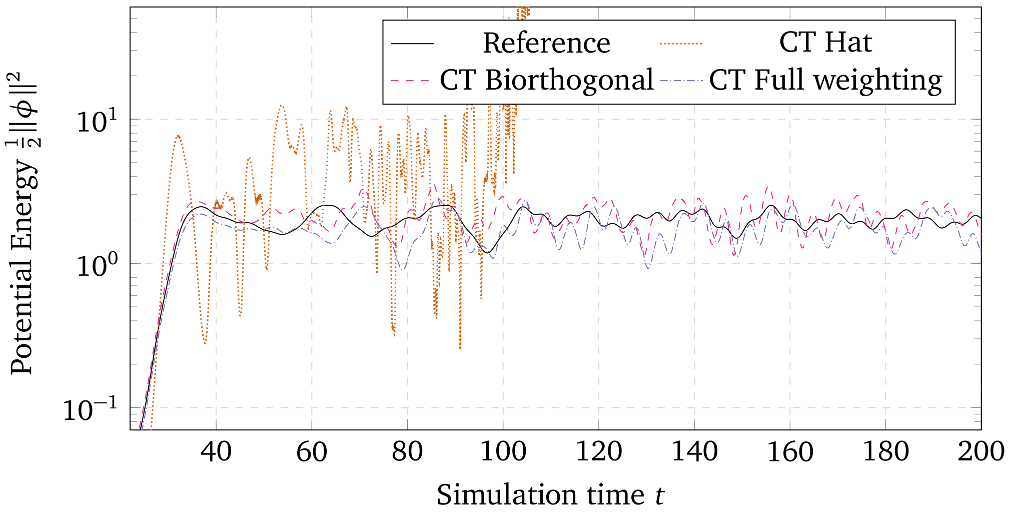


Figure 3.10.: Potential energies on the coarsest component grid with $\vec{\ell} = \vec{\ell}^{\text{min}}$ in the two-stream instability scenario. (Figure first published in [137])

Looking at a single grid's energy curves in Figure 3.10, it makes sense why the ψ^{hat} CT simulation becomes unstable: The coarsest component grid's energy curve already shows large deviations from the reference solution during the linear phase of the simulation, resulting in high oscillations right away. This component grid never captures the behavior of the combined solution, in stark contrast to the mass-conserving CT. Knowing this, it is not at all surprising to see the simulation with hierarchical hat functions become unstable; What appears more surprising is that the combined energy in Figure 3.9 nicely matches the prediction up until $t \approx 70$. This, again, is the result of the CT's error cancellation, cf. Section 2.3.1.

With the mass-conserving CT, the component grid solutions follow the reference solution reasonably well and display energy levels close to their respective combined solutions. One can observe a rather uniform oscillation of similar frequency with a larger amplitude in the medium-resolution full grid solution compared to the mass-conserving combination solutions. One may therefore evaluate it slightly inferior to the ones obtained with a full weighting or a biorthogonal CT, at the same amount of main memory spent.

From these figures, it can be inferred that the two expectable pitfalls—too high oscillations and (artificial) damping of the electric energy—do not impede the mass-conserving CT's accuracy. However, when performing multiple solver steps per combination step, even the mass-conserving CT would become unstable some time into the nonlinear phase for a range of setups tested (different solver and combination time steps). This matches the limitations discussed in Section 3.4.4, as multiple solver time steps allow the advection direction to be changed within the same combination time step. The problem may potentially be alleviated by adapting the combination interval to the changes in the advection terms on the component grids.

Coming back to the topic of conservation of mass in the (per-se conserving) SeLaLib simulations, Figure 3.11 shows the mass trajectories of the combined solutions. The largest deviation to the expected result ($\approx 1 \times 10^{-9}$) is introduced by the initial interpolation. Otherwise, the mass for the combination with $\psi^{\mathtt{hat}}$ is conserved up to a level of less than 1×10^{-9} prior to the instability occurring. For $\psi^{\mathtt{bo}}$ and $\psi^{\mathtt{fw}}$, the combined mass is conserved at least up to the output accuracy of 1×10^{-11}. This is in close agreement with the result seen in Section 3.4.2.

Figure 3.12 shows the evolution of mass on the coarsest component grid, whose electric energy was plotted in Figure 3.10. Like for the advection experiment, the mass-conservation property of $\psi^{\mathtt{bo}}$ and $\psi^{\mathtt{hat}}$ can be observed, while $\psi^{\mathtt{hat}}$ shows large changes in mass on the coarsest component grid (more than 0.1 even at simulation times $t < 60$ when the combined energy still looks reasonable and the combined mass is still conserved up to 2×10^{-9}). The oscillations for $\psi^{\mathtt{hat}}$ in Figure 3.11 in the order of 1×10^{-10} can be attributed to numerical error accumulation as the mass is 'shifted' between the 84 component grids. The combined mass remains at this value until after time ≈ 94 where the values of the potential energy start to

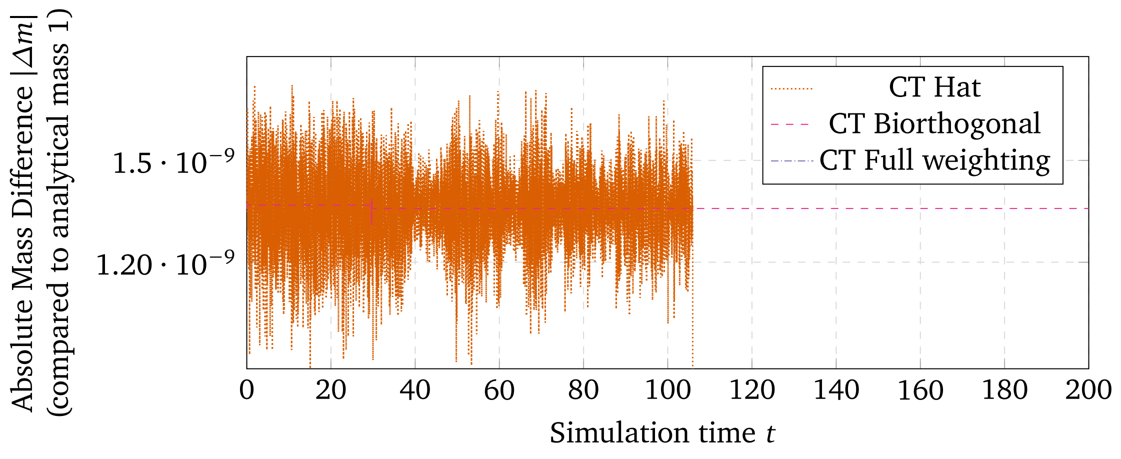


Figure 3.11.: Difference between the combined mass $m_{\mathrm{ct}}(t) = \sum_{\vec{\ell} \in \mathcal{I}^{\mathrm{CT}}} \lambda_{\vec{\ell}} \cdot \int f_{\vec{\ell}}(x,t)\,dx$ and the analytical mass $m = 1$ over time in the two-stream instability scenario. The curves of the two mass-conservig functions exactly ovelap. The largest difference ($\approx 1 \times 10^{-9}$) is introduced by the initial interpolation. Note the logarithmic scale for the vertical axis. (Figure first published in [137])

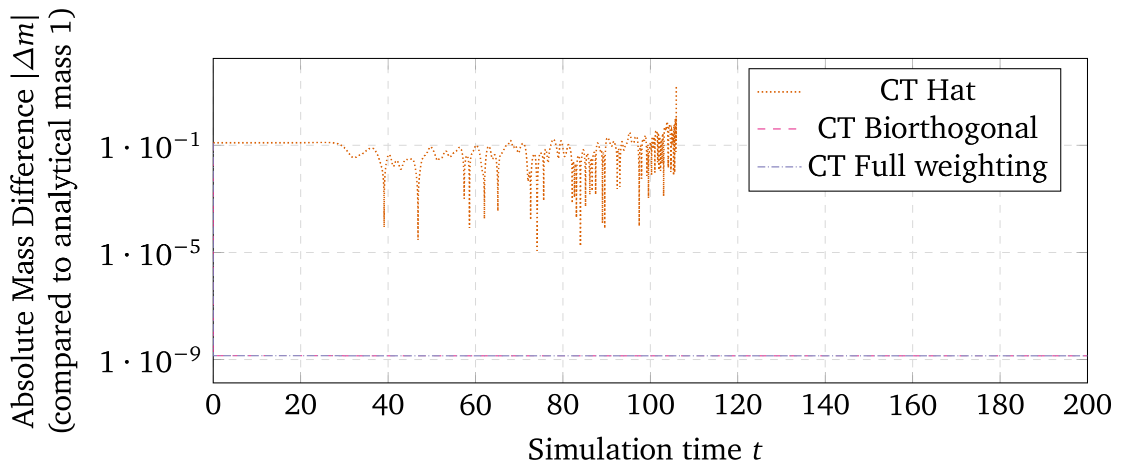


Figure 3.12.: Difference between the mass on a single component grid $m_{\vec{\ell}}(t)$ with $\vec{\ell} = \vec{\ell}^{\min} = (3,3,3,3,3,3)$ and the analytical mass $m = 1$ over time in the two-stream instability scenario. Note the logarithmic scale for the vertical axis. (Figure first published in [137])

oscillate strongly. This matches the expectation that the error cancellation should work better for a linear quantity such as mass than for the nonlinear quantity of the potential energy (quadratic in f).

This numerical plasma two-stream instability experiment hints that the 'small scale' observation from Section 3.3.2 holds: The conservation of mass and the stability of the simulation are two sides of the same coin. Both can be considered a consequence of the regularity of the dual scaling function, cf. Section 2.1.2.

3.6. Practical Assessment of Standard and Mass-Conserving Basis Functions

While the theoretical properties of the hierarchical hat basis were discussed in Section 2.1.2, this section addresses the observations and the technical details in the simulation experiments.

Conservation, Accuracy, Stability

The function spaces are the same for all considered basis representations—piecewise linear functions—and all transforms are unique and lead to perfect reconstruction (up to machine precision) if inverted. In fact, the combined sparse grid representation f^{CT} will be exactly the same after a single time step, regardless of the hierarchical basis functions used. It was illustrated in Section 3.3 that the decombination into the component grids is the critical point where the different basis functions show their differences.

In particular, while the *combined* mass is conserved also for $\psi^{\texttt{hat}}$, cf. Figure 3.11, mass conservation on the *component grids* is only asserted for $\psi^{\texttt{bo}}$ and $\psi^{\texttt{fw}}$, cf. Figures 3.5 and 3.12.

Furthermore, if the 'true' solution to a problem is not very regular, then the mass-conserving basis functions can be expected to be more accurate than the standard hat functions, cf. Section 2.1.2. This was observed in practice for the advection experiment in Figure 3.6, where the approximated function is only in C^0. In this case, the sparse grid projection error of the final simulation solution could be reduced by a factor of ≈ 3.

Finally, the stability of the simulation is directly related to the stability of the multiscale bases. Like mentioned in Section 2.1.2, the lacking regularity of the dual wavelet of $\psi^{\texttt{hat}}$—the Dirac delta distribution—leads to an unstable basis. This is reflected in the DisCoTec + SeLaLib simulations becoming unstable with $\psi^{\texttt{hat}}$, cf. Figures 3.8 and 3.9. Interestingly, the numerical instability is hinted at much earlier from the quantities of interest on single component grids, cf. Figure 3.10, but due to the CT's error cancellation property, the individual grids correct for each other over long stretches of simulation time in the combined solution. Conversely, the regularity of $\psi^{\texttt{bo}}$ and $\psi^{\texttt{fw}}$, which are each other's dual wavelets, leads to a stable basis. Accordingly, the same simulations that became unstable with $\psi^{\texttt{hat}}$ remain stable with $\psi^{\texttt{bo}}$ and $\psi^{\texttt{fw}}$.

This is an important observation, since numerical instability is a commonly encountered problem in standard combination technique simulations, cf. [102, 172].

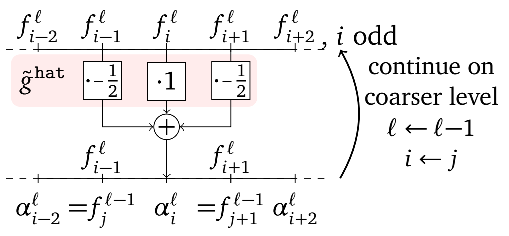
Computational Cost

The complexity of computing the basis transforms is $\mathcal{O}(N)$ in all cases, and no extra memory needs to be allocated for the transform, as all computation can be performed in-place. This can be seen from the coefficients in Section 2.1.1. The coefficients result in (one or two) data sweeps per level along the hierarchization dimension, illustrated by Figure 3.13.

However, the true computational cost of using the mass-conserving basis functions in place of the standard hat functions depends on the parallelization: If the data along the considered dimension is local to a process, then the mass-conserving basis transforms can be expected to take approximately twice as long as the standard hat function transforms—potentially much less if cache blocking [70] is used. But if the data needs to be communicated between processes, $\psi^{\texttt{hat}}$ has a significant performance benefit: Due to its interpolating property, only $\mathcal{O}(\log N)$ data needs to be communicated between neighboring processes prior to computation [74]. For $\psi^{\texttt{bo}}$ and $\psi^{\texttt{fw}}$, the entire data may have to be communicated (in the case that the minimum hierarchization level is 0), which poses a significant overhead. Maybe even worse, this communication has to take place with all processors that hold data along this particular data axis. (A run-time comparison for the different basis functions is given in Appendix A.1, which reflects this behavior.)

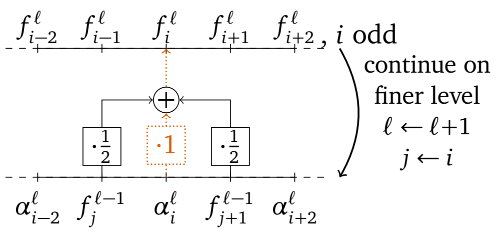
An alternative would be to interlace computation with communication by exchanging the coefficients required from other processes after every level update (i. e., after every loop iteration in Figures 3.13a to 3.13f respectively). While this would reduce the overall communication volume to $\mathcal{O}(\log^2 N)$, the expected run time due to the communication latencies would still be significant.

Overall, through these considerations one of the strengths of the CT approach becomes apparent: There is a clear separation between the CT and the solver, which means that the solver can be developed independently of any multiscale considerations. Vice versa, as long as the solver supports nested discretizations, the CT can be used without any changes to the solver, and basis functions different from $\psi^{\mathtt{hat}}$ can be used in a straightforward manner, such as to improve the conservation, accuracy, and stability properties of the scheme.



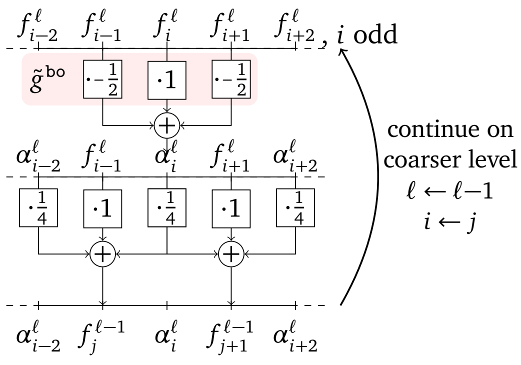
f^ℓ_{i-2} f^ℓ_{i-1} f^ℓ_i f^ℓ_{i+1} f^ℓ_{i+2}, i odd
$\tilde{g}^{\mathtt{hat}}$
$\cdot -\frac{1}{2}$
$\cdot 1$
$\cdot -\frac{1}{2}$
continue on coarser level
$\ell \leftarrow \ell-1$
$i \leftarrow j$
$\alpha^\ell_{i-2} = f^{\ell-1}_j$ α^ℓ_i $= f^{\ell-1}_{j+1}$ α^ℓ_{i+2}

(a) Hierarchical hat basis: For the hierarchization, new values on the coarser level are identical to the previous ones.



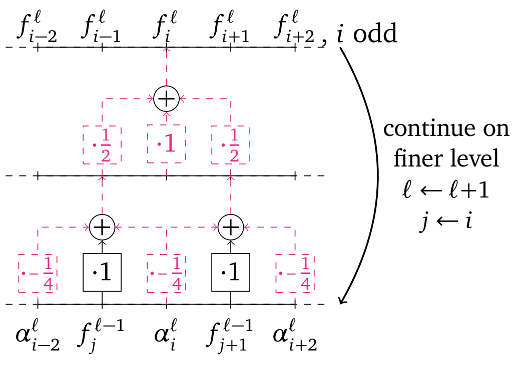
f^ℓ_{i-2} f^ℓ_{i-1} f^ℓ_i f^ℓ_{i+1} f^ℓ_{i+2}, i odd
$\cdot\frac{1}{2}$
$\cdot 1$
$\cdot\frac{1}{2}$
continue on finer level
$\ell \leftarrow \ell+1$
$j \leftarrow i$
α^ℓ_{i-2} $f^{\ell-1}_j$ α^ℓ_i $f^{\ell-1}_{j+1}$ α^ℓ_{i+2}

(b) Hierarchical hat basis: Coarser values are kept, finer values are obtained by the sum of interpolation and hierarchical surplus.



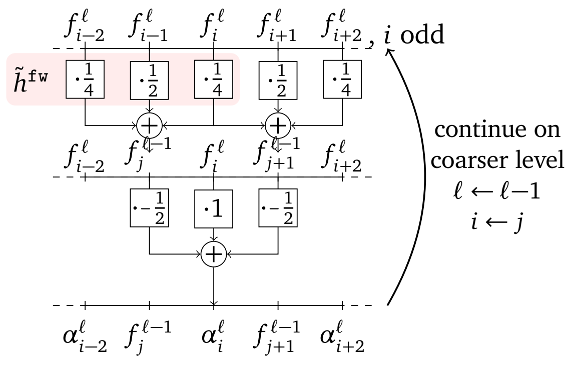
f^ℓ_{i-2} f^ℓ_{i-1} f^ℓ_i f^ℓ_{i+1} f^ℓ_{i+2}, i odd
$\tilde{g}^{\mathtt{bo}}$
$\cdot -\frac{1}{2}$
$\cdot 1$
$\cdot -\frac{1}{2}$
α^ℓ_{i-2} f^ℓ_{i-1} α^ℓ_i f^ℓ_{i+1} α^ℓ_{i+2}
$\cdot\frac{1}{4}$ $\cdot 1$ $\cdot\frac{1}{4}$ $\cdot 1$ $\cdot\frac{1}{4}$
continue on coarser level
$\ell \leftarrow \ell-1$
$i \leftarrow j$
α^ℓ_{i-2} $f^{\ell-1}_j$ α^ℓ_i $f^{\ell-1}_{j+1}$ α^ℓ_{i+2}

(c) Biorthogonal basis: Hierarchical coefficients are calculated as before, but then the scaling function coefficients are updated, too.



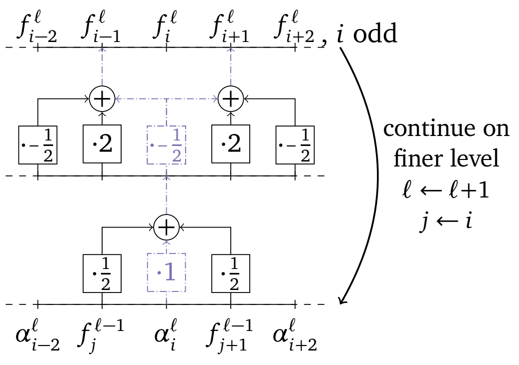
f^ℓ_{i-2} f^ℓ_{i-1} f^ℓ_i f^ℓ_{i+1} f^ℓ_{i+2}, i odd
$\cdot\frac{1}{2}$ $\cdot 1$ $\cdot\frac{1}{2}$
$\cdot -\frac{1}{4}$ $\cdot 1$ $\cdot -\frac{1}{4}$ $\cdot 1$ $\cdot -\frac{1}{4}$
continue on finer level
$\ell \leftarrow \ell+1$
$j \leftarrow i$
α^ℓ_{i-2} $f^{\ell-1}_j$ α^ℓ_i $f^{\ell-1}_{j+1}$ α^ℓ_{i+2}

(d) Biorthogonal basis: The pattern of inverse signs for odd-numbered stencil positions compared to the hierarchization is visible.



f^ℓ_{i-2} f^ℓ_{i-1} f^ℓ_i f^ℓ_{i+1} f^ℓ_{i+2}, i odd
$\tilde{h}^{\mathtt{fw}}$
$\cdot\frac{1}{4}$ $\cdot\frac{1}{2}$ $\cdot\frac{1}{4}$ $\cdot\frac{1}{2}$ $\cdot\frac{1}{4}$
f^ℓ_{i-2} $f^{\ell-1}_j$ f^ℓ_i $f^{\ell-1}_{j+1}$ f^ℓ_{i+2}
$\cdot -\frac{1}{2}$ $\cdot 1$ $\cdot -\frac{1}{2}$
continue on coarser level
$\ell \leftarrow \ell-1$
$i \leftarrow j$
α^ℓ_{i-2} $f^{\ell-1}_j$ α^ℓ_i $f^{\ell-1}_{j+1}$ α^ℓ_{i+2}

(e) Full weighting basis: The filters are virtually the same as in the biorthogonal case, but applied in reverse order.



f^ℓ_{i-2} f^ℓ_{i-1} f^ℓ_i f^ℓ_{i+1} f^ℓ_{i+2}, i odd
$\cdot -\frac{1}{2}$ $\cdot 2$ $\cdot -\frac{1}{2}$ $\cdot 2$ $\cdot -\frac{1}{2}$
$\cdot\frac{1}{2}$ $\cdot 1$ $\cdot\frac{1}{2}$
continue on finer level
$\ell \leftarrow \ell+1$
$j \leftarrow i$
α^ℓ_{i-2} $f^{\ell-1}_j$ α^ℓ_i $f^{\ell-1}_{j+1}$ α^ℓ_{i+2}

(f) Full weighting basis: One can observe the 'lifting' property, allowing to greedily compute the transform in-place.

Figure 3.13.: Filters for the hierarchization and dehierarchization operations for the different multiscale bases. The hierarchization (left column) is described by Equations (2.14) and (2.15), the dehierarchization (right column) is described by Equation (2.16). In Figures (c) to (f), filters from Equations (2.18) and (2.19) that are not explicitly depicted arise from the subsequent execution of the two lifting steps. (Figure first published in [137])

Chapter 4

High Performance Computing and the DisCoTec Code

It is a well-established fact that the computational requirements for high-fidelity controlled fusion simulations are enormous. High performance computing (HPC) on massively parallel systems is currently the only way to deal with the necessary memory and run time effort. For instance, both the DOE [20] and the ITER consortium [107] intend to coordinate HPC efforts for controlled fusion energy generation. And conversely, plasma simulations are one of the featured applications in the HPC community's 'exascale roadmap' [35].

The challenging memory requirements were already discussed in Chapter 1; This chapter will investigate how they translate to the current HPC challenges and how they are addressed in the DisCoTec framework, which is the basis for the simulations presented in this thesis.

4.1. When Will We Achieve Exascale Computing?

For more than ten years, the high performance computing (HPC) community has been working towards 'exascale computing' [16, 17, 35]. But as of yet, no single simulation application code has publicly reported exascale performance. What makes exascale and, more generally, HPC so notoriously difficult?

Typical modern HPC architectures consist of many compute nodes, each of which contains several processors with shared memory, and may contain accelerators such as GPUs [153]. In analogy to the analysis by Matsuoka et al. [110], one can define 'exascale' in three different ways, listed by increasing difficulty to attain:

1. *exaop system* refers to any computer theoretically capable of performing 10^{18} operations per second,

2. *exaflop system* refers to any computer capable of achieving over 10^{18} double-precision floating-point operations ('FP64') per second ('flop/s') on the Linpack benchmark, and

3. *exascale system* denotes any computer executing a scientific application with a sustained performance of over 1 exaflop/s in FP64.

The Frontier system is a publicly funded machine that is reportedly the first exaflop system as of 2022 [110]. But looking at the question of an exascale system, current figures with the (more realistic than Linpack) HPCG benchmark are only in the one- to two-digit petaflop range [153].

One of the reasons for this discrepancy is that the architectures are very complex, making it a challenging task to develop efficient codes. After all, the benefits of parallelization, vectorization, cache hierarchies, and other optimizations [70] are only reaped if they can be applied to the algorithm at hand, and even then very specialized skills are required to develop and maintain an efficient program.

The other main reason for the discrepancy between exaflop and exascale is that for real-world applications, the run time of the program will be dominated by memory access and data communication times, not by floating-point operations [9]. Currently, the largest machines rely heavily on GPU accelerators [153]. This makes the imbalance between communication and computation even more severe, because GPUs provide a lot of compute, but the memory link to the node's main CPU memory is relatively slow. As a consequence, in programming for these systems, it is vital to develop algorithms and codes with a special focus on data flow [174], avoiding unnecessary data movement and communication wherever possible.

4.2. Related Work: High Performance Computing for Multiscale PDE Solvers and on Multiple Systems

Besides DisCoTec and its predecessor versions [74, 75, 77, 122], there are only few codes dedicated to the sparse grid combination technique. In particular, while the SG++ suite [124, 158] that DisCoTec originated from does provide functionality for combination technique methods, it has a stronger focus on adaptive optimization [162], uncertainty quantification [142], and machine learning applications [146] with the hierarchical sparse grid representation.

The Sparse Grids Matlab Kit [126, 127] by Piazzola and Tamellini is targeted towards teaching and uncertainty quantification with the CT. It provides a user-friendly MATLAB interface for the CT, and also dimensional adaptivity [53] for nested and non-nested sequences of component grid collocation points. The code's parallelism is limited to shared memory only, as it does not target HPC applications. The same holds true for the sparseSpACE code [120], which provides an implementation for automatic spatial adaptivity in the CT for integration [121], interpolation, uncertainty quantification, machine learning methods, and PDE solvers.

Both the hierarchical construction and the combination technique have been successfully used for denoising massively parallel Particle-In-Cell solvers [31, 115, 145].

To encounter PDE solvers with similarity to DisCoTec's application, one has to look at the wider scope of multiscale methods. In this realm, there are two approaches that should be highlighted: Multigrid and wavelet solvers. As a multigrid example, the solver HyTeG is used in the TerraNeo project [7, 88, 159] for earth mantle convection problems, and uses adaptively refined three-dimensional tetrahedral meshes for discretization. To the author's best knowledge, TerraNeo also holds the record for the mesh-based simulation with the highest number of DOF to date with 1.1×10^{13} DOF [7, 159].

For solvers directly based on (lifting) wavelets, the binary black hole merger simulations using Dendro-GR by Fernando et al. [45, 46] are a good HPC example. Notably, Dendro-GR uses interpolets of orders up to eight for a finite-difference solution to the Einstein equations. These interpolets live on octree structures (which is somewhat contradictory to the fact that 'sparse grid' is mentioned [46, p.4]. Supposedly, the authors of Dendro-GR were not aware of sparse grid terminology). Using this approach, Dendro-GR was demonstrated to scale simulations to up to 229,376 cores, and also employ GPGPU performance [44].

The wavelet solvers mentioned in Section 3.1 exhibit only limited parallelism, and are therefore not considered here.

In a spirit similar to the widely-distributed CT, which is discussed in Section 4.5, the HPC community has been exploring various approaches for coupling multiple systems into a single meta-system ('metacomputing' or 'grid computing') [48]. These approaches typically require a tight coupling between systems. However, this has nowadays become impossible to provide for publicly funded HPC infrastructure because of (justified!) security concerns. But many of the approaches developed for grid computing have found their way into modern—typically proprietary—cloud infrastructures after all.

In 2014, the terabit demonstrator project [13] showcased that high bandwidths between publicly funded systems are achievable, if sufficient investment is provided by compute centers and infrastructure providers.

One metacomputing approach particularly related to the widely-distributed CT is PACX-MPI [8]. Its aim was to provide a 'normal' MPI interface for applications that are distributed across multiple systems. Within the run time, communication daemons manage application startup, MPI communication, and, if necessary, data conversion between the different systems. Operations within the same system would be mapped to the native MPI library at little to no overhead. Operations across systems would be translated into PACX-MPI requests, which are then transferred over the external network to the receiving daemon on the target system, where they are translated back into native MPI operations. This is very similar to the TCP variant of the widely-distributed CT. However, PACX-MPI requires direct access via open network ports between the systems, which is typically blocked nowadays because of the aforementioned increased security requirements—the same ones that also pose a challenge for the widely-distributed CT.

4.3. DisCoTec Software Architecture and Parallelism

The framework DisCoTec [151] was originally derived by Heene [74, 75] from the sparse grid toolbox SG++ [124, 158]. Since then, it has grown into its own MPI-based [113] highly parallel framework for combination technique simulations. It was open-sourced in 2019 under the GNU Lesser General Public License [49]. The acronym DisCoTec for the open-source code stands for **dis**tributed **co**mbination **tec**hnique[1]. To use a PDE solver with DisCoTec, the solver needs to expose an interface to pass the MPI communicator to the solver (to allow for multiple solver instances running at the same time); Additionally, it has to provide functions to trigger initialization and time stepping of the respective instance. All data exchange between the component simulation instances is handled by DisCoTec. DisCoTec assigns ranks to be *worker processes*, and (optionally) one rank to be the *manager process*. For the workers, DisCoTec employs two modes of parallelization: first, the

[1]Naming credit goes to Michael Obersteiner.

domain-decomposition based Cartesian parallelism provided by the solver itself, and second, the additional parallelism enabled by the CT's independent grid time steps.

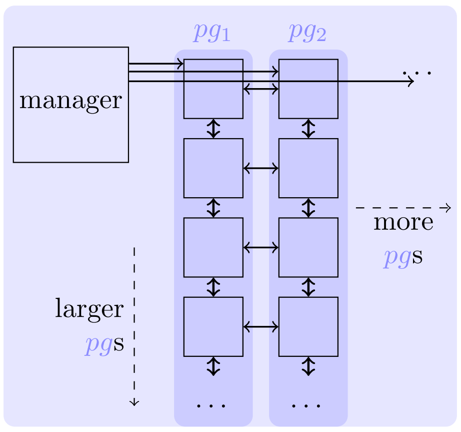


Figure 4.1.: DisCoTec's MPI communication set-up: Each black square denotes an MPI rank. Ranks are bundled into process groups pg_i and process groups are collectively assigned tasks, i. e., component grids and the pertaining solver operators. We have traditional parallelism based on Cartesian domain decomposition within the process groups, and the CT's added parallelism between the process groups. The manager rank is optional.

Each component grid is typically distributed among many processes by Cartesian domain decomposition, which together form a process group pg_i. And vice versa, every process group will typically have multiple component grids to work on, which allows for load balancing on the level of component grids [76, 135], as illustrated by Figure 4.2. Thus, the number of processes in a given simulation can be increased either by making the process groups larger, or by adding more process groups (cf. extending to the bottom or right in Figure 4.1). In the following, the number of process groups is denoted n_g, and the number of processes per group is denoted n_p.

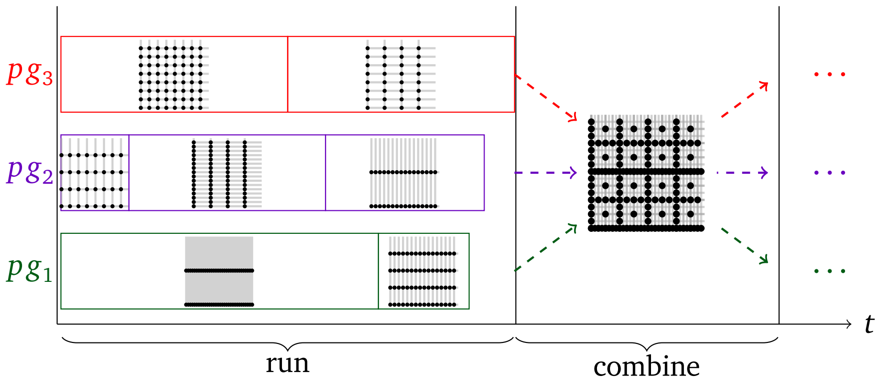


Figure 4.2.: Schematic timeline of DisCoTec time-stepping: The component grids assigned to the process groups perform their solver time steps in parallel, but may take varying amounts of time to complete their time step. This can lead to load imbalance, since the subsequent combination step implicitly synchronizes between the process groups.

Over the course of the presented research, DisCoTec was developed further. A representative list of contributions is given towards the end of this chapter in Section 4.7.

4.4. Communication Schemes and Communication Volumes

Figure 4.1 shows the distribution of MPI ranks in DisCoTec. Each process group shares a set of component grids, together with functions and data structures to perform the time step updates. These *tasks* typically link to the (Vlasov) solver implementation. All of the solver's MPI communication—such as ghost layer exchange and computation of quantities of interest—happens within the process group only (vertically in Figure 4.1).

Within the process group, the same 'traditional' Cartesian domain decomposition is used as in the structured grid solver. The sparse grid data present within an MPI rank also covers the same Cartesian subdomain of Ω. This can pose a load-balancing problem: Note that the j-direction coordinates present on $\ell_j^{\min}$ will be far 'heavier' in terms of DOF than those only present in higher levels. This is a consequence of the sparse grid construction [162, p. 2.4.1]. If one-sided boundaries are used (as is suitable for periodic boundary conditions), one can achieve balanced numbers of DOF between the ranks of the process group by using powers of two as the process group size n_p. Then, the maximum process count in each dimension j is given by $2^{\ell_j^{\min}}$ (the minimum number of grid points in that dimension), and the total rank count per group may be up to ${n_p}^{\max} = 2^{|\vec{\ell}^{\min}|_1}$. If no boundary points or both boundary points are used per dimension, the load balancing within the process group becomes severely more challenging, which is another reason why one-sided boundary points are used in Sections 5.4.1, 5.4.2 and 5.5.

For the remainder of this section, it will be discussed how exactly the DisCoTec implementation uses MPI communication for the different steps in the time-stepping combination, as outlined in Algorithm 2.1.

Initialization and solver time step updates can be performed embarrassingly parallel between the process groups; If a manager rank is present, it can be used to achieve dynamic load balancing based on the run time of the first solver time step [76, 135].

The default rank placement is designed to put ranks of the process groups close to each other, to minimize the communication times for the (typically more run-time intense) within-group vertical communication. This means that the first n_p ranks, which form the first process group, are placed close to each other by starting to fill the first, second,... allocated node, then the second process group starts, and so on.

Since the combination needs to take place in the hierarchical basis representation, all full grids are hierarchized individually, still through vertical communication in Figure 4.1, within the process group. After hierarchization and collection of weighted values according to Equation (2.26) in the sparse grid data structure, the data needs to be communicated horizontally to be all-reduced between the process groups. After this, the sparse grid data can be scattered back to the tasks that depend on the respective subspace, and dehierarchization takes place, before the next time step continues the progression of simulation time. Algorithm 4.1 illustrates this by expanding upon Algorithm 2.1 and adding arrows ($\longleftrightarrow$ and $\updownarrow$) to denote horizontal and vertical communication, respectively.

Process groups are the main tool how DisCoTec avoids global communication and synchronization: All communication has to take place in either of the directions, vertical or horizontal, but never through `MPI_COMM_WORLD`. In addition to MPI's distributed-memory parallelism [113], DisCoTec can employ shared-memory parallelism with OpenMP [123], where the number of OpenMP threads per rank is denoted n_t. This parallelism can be leveraged if the structured grid solver uses OpenMP as well. Note that trading MPI ranks for higher numbers of OpenMP threads is not necessarily more performant and can lead to undesired cache effects, as will be discussed in Section 5.4.

Algorithm 4.1 Time-Stepping CT

```
1:  procedure TimeSteppingCT($\mathcal{I}^{\text{CT}}$, $\mathcal{I}^{\text{SG}}$)
2:      for $t \leftarrow 0, N_{\text{steps}}$ do
                                                                                    run
3:          for all $\vec{\ell} \in \mathcal{I}^{\text{CT}}$ do
4:              solve $\vec{\ell}$      ▷ Update component grids from time step $t$ to $t+1$      $\updownarrow$
5:          end for
                                                                                    combine
6:          for all $\vec{\ell} \in \mathcal{I}^{\text{CT}}$ do
7:              hierarchize $\vec{\ell}$      ▷ Basis transform: nodal → hierarchical      $\updownarrow$
8:          end for
9:          Reduce($\mathcal{I}^{\text{CT}}$, $\mathcal{I}^{\text{SG}}$)
       ▷ Reduce hierarchical coefficients to sparse grid with Equation (2.26)      $\longleftrightarrow$
10:         Scatter($\mathcal{I}^{\text{CT}}$, $\mathcal{I}^{\text{SG}}$)
                ▷ Scatter hierarchical coefficients to component grids      $\longleftrightarrow$
11:         for all $\vec{\ell} \in \mathcal{I}^{\text{CT}}$ do
12:             dehierarchize $\vec{\ell}$      ▷ Basis transform: hierarchical → nodal      $\updownarrow$
13:         end for
14:     end for
15: end procedure
```

The DisCoTec communication for the vertical (de)hierarchization steps for hierarchical hat functions was extensively discussed by Heene [74, section 3.3]. For the horizontal reduction operations between the process groups, Hupp et al. [85, 87] investigated data sizes and scaling behavior. They, however, assumed that the component grids in the combination scheme are not distributed, but are held by a single node only, and conversely, that each node would only hold a single component grid. Still, the analysis is useful for the current, more distributed setting, and this thesis uses the same or similar notation as Hupp et al. [85] to discuss the data volumes in the reduction step:

- $H_{\vec{\ell}}$: the (subspace) grid data structure corresponding to the hierarchical increment space $W_{\vec{\ell}}$. Typically, this means a vector of hierarchical coefficients α.

- $C_{\vec{\ell}}$: the (component) full grid data structure used to represent the anisotropic full grid space $V_{\vec{\ell}}$. If only a subset of the data corresponding to a hierarchical subspace $\vec{\ell}'$ is meant, it is denoted by the index notation $C_{\vec{\ell}}[\vec{\ell}']$.

- $\mathbf{SG}_{\mathcal{I}^{\mathrm{SG}}}$: the grid corresponding to the sparse grid space defined by the hierarchical index set $\mathcal{I}^{\mathrm{SG}}$.

Then, $\left|C_{\vec{\ell}}\right|$ is the number of grid points in the component grid, $\left|H_{\vec{\ell}}\right|$ is the number of grid points in the hierarchical grid, and the respective number of component grids and subspaces in the combination scheme is given by $\left|\mathcal{I}^{\mathrm{CT}}\right|$ and $\left|\mathcal{I}^{\mathrm{SG}}\right|$, respectively (see Sections 2.2 and 2.3 for the definitions of the index sets). The set of component grids $\mathbf{CG}(H_{\vec{\ell}})$ that contain the hierarchical space $W_{\vec{\ell}}$ is given by all component grids with equal or higher level [85]:

$$\mathbf{CG}(H_{\vec{\ell}}) := \left\{C_{\vec{\ell}'} \forall \vec{\ell}' \in \mathcal{I}^{\mathrm{CT}} : H_{\vec{\ell}} \subseteq C_{\vec{\ell}}\right\} = \left\{C_{\vec{\ell}'} \forall \vec{\ell}' \in \mathcal{I}^{\mathrm{CT}} : \vec{\ell}' \geq \vec{\ell}\right\}. \tag{4.1}$$

The notation generally assumes the vector comparisons to be true iff they are true for each element, for instance

$$\vec{\ell}' \geq \vec{\ell} := \forall j = 1 \ldots d : \ell'_j \geq \ell_j. \tag{4.2}$$

The sparse grid, on the other hand, consists of the union of all occurring hierarchical spaces

$$\mathbf{SG}_{\mathcal{I}^{\mathrm{SG}}} = \bigcup_{\vec{\ell} \in \mathcal{I}^{\mathrm{SG}}} H_{\vec{\ell}}, \text{ with } \mathcal{I}^{\mathrm{SG}} := \left\{\vec{\ell}' \in \mathcal{I}^{\mathrm{CT}} : H_{\vec{\ell}'} \neq \emptyset\right\}. \tag{4.3}$$

The sparse grid index set will be downward closed by construction, i. e.,

$$\forall \vec{\ell} \in \mathcal{I}^{\mathrm{SG}} : \vec{\ell}' \leq \vec{\ell} \implies \vec{\ell}' \in \mathcal{I}^{\mathrm{SG}}, \tag{4.4}$$

and one can denote the number of DOF in the sparse grid by $|\mathbf{SG}_{\mathcal{I}^{\mathrm{SG}}}|$.

For a set of component grid levels $A \subseteq \mathcal{I}^{\mathrm{CT}}$, the grid points shared with all other levels in the combination scheme $\mathcal{I}^{\mathrm{CT}} \setminus A$ can be computed by finding all hierarchical spaces that are present in A, but not A alone:

$$\mathbf{sharedG}(A) := \bigcup_{\vec{\ell}' \in A} C_{\vec{\ell}'} \cap \bigcup_{\vec{\ell}' \in (\mathcal{I}^{\mathrm{CT}} \setminus A)} C_{\vec{\ell}'} = \bigcup_{\substack{H_{\vec{\ell}} \in \mathcal{I}^{\mathrm{SG}}: \\ \mathbf{CG}(H_{\vec{\ell}}) \cap \{C_{\vec{\ell}'} \forall \vec{\ell}' \in A\} \neq \emptyset, \\ \mathbf{CG}(H_{\vec{\ell}}) \not\subseteq \{C_{\vec{\ell}'} \forall \vec{\ell}' \in A\}}} H_{\vec{\ell}}. \tag{4.5}$$

For across-group parallelism, this is the minimum amount of data that has to be communicated for a process group holding a subset A of the combination scheme. The minimum data volume in bytes is generally proportional to $|\mathbf{sharedG}(A)|$, and depends on the exact assignment of the component grids to process groups. What exactly constitutes a 'good' or 'bad' assignment will be discussed in Section 4.6.

There are four reduction schemes currently implemented in DisCoTec, the first two of which closely resemble the ones in [85, 87]. However, there are differences: Since DisCoTec uses distributed-memory parallelism, the implementation needs to differ from the original description, and MPI as well as main memory limitations come into play. For instance, Hupp et al. [85] assume a subspace-wise reduction, in which the reduce buffer is allocated, filled, and reduced for each subspace separately. This becomes unfeasible as the number of subspaces becomes very high, as it would imply too many calls to MPI library functions. Attempting to address this with concurrent reductions leads to erroneous behavior, since for `MPI_Allreduce` or `MPI_Iallreduce`, the ordering of messages is not guaranteed for the same communicator (the—even more unfeasible—alternative would be to provide separate communicators for every subspace). Conclusively, DisCoTec stores the sparse grid data as a sequential vector of accumulated hierarchical values α, which is indexed by the level $\vec{\ell}$ that each sequence represents. A practical feature is that the allreduce operations can be chunked to lower the memory footprint—without chunking, the memory allocated for the buffer may have to be at least doubled in size for the reduction operation. DisCoTec further avoids the use of the non-blocking `MPI_Iallreduce` operation, since unnecessarily large temporary allocations were observed for various MPI implementations, cf. Appendix A.2. However, it will make sense to revisit this decision once the commonly-used MPI implementations have improved in this regard, or for settings where memory usage is not a concern.

One optimization mentioned in [85] is also commonly used in DisCoTec: The highest diagonal of the sparse grid, i. e., the set of finest-resolved hierarchical spaces, can always be omitted from the reduction operation, since they are relevant for a single full grid only ($|\mathbf{CG}(H_{\vec{\ell}})| = 1$) and need not be reduced at all.

The four reduction schemes currently available in DisCoTec are briefly reviewed in the following.

4.4.1. Sparse Grid Reduce

The sparse grid reduce algorithm performs a maximum reduction at the beginning of the program to determine the sizes of the different subspaces $H_{\vec{\ell}}$. Each process group then allocates those space sizes, regardless of whether they are actually needed by the group. This (potentially very long) indexed vector of hierarchical coefficients is allreduced in every combination step, in a parameterizable chunk size. Algorithm 4.2 gives an overview of the algorithm in pseudocode.

Algorithm 4.2 Sparse Grid Reduce Algorithm

```
1:  procedure SGReduce($\mathcal{I}^{CT}$, $\mathcal{I}^{SG}$)
                                                              LocalReduce
2:      for all $\vec{\ell}' \in \mathcal{I}^{SG}$ do
3:          $H_{\vec{\ell}'} = \vec{0}$
4:      end for
5:      for all $\vec{\ell} \in \mathcal{I}^{CT}$ do
6:          for all $\vec{\ell}' \in \mathcal{I}^{SG}$ do
7:              if $C_{\vec{\ell}} \in \mathbf{CG}(H_{\vec{\ell}'})$ then
8:                  $H_{\vec{\ell}'} \mathrel{+}= c^c_{\vec{\ell}} \cdot C_{\vec{\ell}}[\vec{\ell}']$      ▷ Locally reduce hierarchical coefficients
9:              end if
10:         end for
11:     end for
12:     AllReduce$_{n_g}$($\mathbf{SG}_{\mathcal{I}^{SG}}$) ▷ Reduce all hierarchical coefficients across all groups
                                                              LocalScatter
13:     for all $\vec{\ell} \in \mathcal{I}^{CT}$ do
14:         for all $\vec{\ell}' \in \mathcal{I}^{SG}$ do
15:             if $C_{\vec{\ell}} \in \mathbf{CG}(H_{\vec{\ell}'})$ then
16:                 $C_{\vec{\ell}}[\vec{\ell}'] = H_{\vec{\ell}'}$      ▷ Scatter hierarchical coefficients
17:             end if
18:         end for
19:     end for
20: end procedure
```

Of all the algorithms presented here, this is the most straightforward one to model, as the communication volume is given by $\frac{|\mathbf{SG}_{\mathcal{I}^{\mathrm{SG}}}|}{n_p}$, independent of which component grid is computed by which group. The transfer time in a latency-dominated setting (assuming infinite bandwidth) is in the order of $\mathcal{O}(\lceil \frac{|\mathbf{SG}_{\mathcal{I}^{\mathrm{SG}}}|}{n_p s_c} \rceil \cdot \log n_g)$, where s_c is the chunk size and n_g is the number of process groups.

4.4.2. Subspace Reduce

The subspace reduce algorithm is in some ways the opposite of the sparse grid reduce algorithm: Only the subspaces $H_{\vec{\ell}}$ that are actually used by the group are allocated. To organize the transfer, the group needs to know which subspaces are needed by which other groups. Therefore, a binary-or reduction is performed on an auxiliary vector to determine the ownership of the subspaces, where each group—with group number i—sets the i-th bit of entry k to 1 if it needs the subspace k, and to 0 otherwise. Accordingly, the numbers in the resulting vector can be used to create communicators for the groups that need the same subspaces; The same number in the vector denotes that it is the same subset of groups $\mathcal{S}$. These communicators are then used to perform the allreduce operations in the combination step, as stated in Algorithm 4.3.

Algorithm 4.3 Subspace Reduce Algorithm

```
1: procedure SubspaceReduce($\mathcal{I}^{\mathrm{CT}}$, $\mathcal{I}^{\mathrm{SG}}$)
2:     LocalReduce($\mathcal{I}^{\mathrm{CT}}$,$\mathcal{I}^{\mathrm{SG}}$)
3:     for all $\mathcal{S}$ do
4:         for all $\vec{\ell}'$ shared by the groups in $\mathcal{S}$ do
5:             $H_{\vec{\ell}'} \mathrel{+}= c^c_{\vec{\ell}} \cdot C_{\vec{\ell}}[\vec{\ell}']$ ▷ Reduce hierarchical coefficients across groups in $\mathcal{S}$
6:         end for
7:     end for
8:     LocalScatter($\mathcal{I}^{\mathrm{CT}}$,$\mathcal{I}^{\mathrm{SG}}$)
9: end procedure
```

The communication volume will be optimal for each group, equal to $\frac{|\mathbf{sharedG}(A)|}{n_p}$, but the resulting communication time depends heavily on the assignment of component grids to process groups. It can potentially be reduced by reordering the communicators such that disjoint subsets of the process groups can communicate concurrently [86].

At this stage, the inherent drawback of this algorithm becomes apparent: The potential number of communicators is in the order of $\mathcal{O}(n_p \cdot 2^{n_g})$, which is infeasible for large numbers of groups, since each communicator will require some main memory. As of 2011, this memory footprint could be as high as 20% of the system memory for 32 communicators, and at the same time, the number of communicators per program was limited to 8192 [6]. Although modern MPI implementations allow for more communicators and also may require less memory per communicator, the impact on the memory footprint of the program is still significant. It becomes a limiting factor for the number of groups, even with very ‘good’ assignments of component grids to process groups. As a consequence, this algorithm is not employed for the experiments in this thesis, but it may be of interest for settings with lower rank counts, and in particular with lower group counts.

4.4.3. Outgroup Sparse Grid Reduce

The outgroup sparse grid reduce algorithm was designed to strike a balance between the two previous algorithms: using only one reduction communicator while avoiding the unnecessary communication of subspaces. Its name is loosely inspired by the sociological concept of outgrouping [155]: Each group tries to keep outwards communication to a minimum, but if data must be communicated, the other groups are not distinguished and the data is shared with all other groups equally.

To achieve this, the subspace ownership is determined with the same bitwise-or reduction as in subspace reduce. But instead of creating a communicator for each subset of groups, each process only determines if a subspace is present on more than one group. If it is, the subspace needs to be allocated on *all* groups. In addition, each group allocates the data that is only needed by itself (the ‘ingroup’ data). The set of ‘outgroup’ hierarchical grids is denoted **outgroupG**, and can be more formally defined as

$$\mathbf{outgroupG} := \bigcup_{i \in 1,\ldots,n_g} \mathbf{sharedG}(A_i), \tag{4.6}$$

where the set of component grid levels A_i is assigned to group number i.

The resulting reduction procedure is summarized as Algorithm 4.4.

The ‘ingroup’ part of the data $\mathbf{SG}_{\mathcal{I}^{\text{SG}}} \setminus \mathbf{outgroupG}$ does not need to be communicated to the other groups, but it is still allocated on the single group to allow for reductions of data on hierarchical subspaces relevant to this group alone.

Algorithm 4.4 Outgroup Sparse Grid Reduce Algorithm

```
1: procedure OutgroupSGReduce($\mathcal{I}^{\text{CT}}$, $\mathcal{I}^{\text{SG}}$)
2:     LocalReduceOSGR($\mathcal{I}^{\text{CT}}$,$\mathcal{I}^{\text{SG}}$)
3:     for all $H_{\vec{\ell}'} \in$ outgroupG do
4:         $H_{\vec{\ell}'} \mathrel{+}= c^c_{\vec{\ell}} \cdot C_{\vec{\ell}}[\vec{\ell}']$
           ▷ Reduce hierarchical coefficients across all groups, some may still be 0
5:     end for
6:     LocalScatter($\mathcal{I}^{\text{CT}}$,$\mathcal{I}^{\text{SG}}$)
7: end procedure
```

The allocated and communicated data sizes, again, depend heavily on the assignment of component grids to process groups, but will be somewhere between the sizes of the sparse grid reduce and the subspace reduce algorithm. This is illustrated by two notable corner cases. First, in the case of two process groups, outgroup sparse grid reduce will be equivalent to subspace reduce for the same component grid assignment. Second, if the assignment of component grids to process groups is so 'bad' that no group has ingroup subspaces—for instance because the main diagonal component grids are distributed in a round-robin fashion—outgroup sparse grid reduce will be equivalent to sparse grid reduce.

4.4.4. Chunked Outgroup Sparse Grid Reduce

When using DisCoTec, some previous publications [119, 138] assumed the necessity of allocating the whole sparse grid at once for the reduction operation, which imposed a considerable memory requirement for larger-scale scenarios. However, looking closely at the data dependencies, this is not strictly necessary. Instead of allocating all the hierarchical subspace data $H \in \mathbf{SG}_{\mathcal{I}^{\text{SG}}}$ at once, it is possible to work on partitions $\mathcal{H}$ of the sparse grid $\mathbf{SG}_{\mathcal{I}^{\text{SG}}}$. Only a subset $\mathcal{H}_k$ of subspaces $H_{\vec{\ell}}$ needs to be allocated at a given time. In DisCoTec, the maximum size $s_c$can be passed as a parameter (in MiB per thread): $|\mathcal{H}_k| \leq s_c \cdot n_t$ (n_t is the number of OpenMP threads per MPI rank). Although it is possible to use smaller chunk sizes for the across-group allreduce than for partitioning into $\{\mathcal{H}\}$, one can assume the same chunk size for both.

When using outgroup sparse grid reduce, the subspace partitions $\mathcal{H}$ are sorted into either outgroup or ingroup subsets. For the ingroup subsets $\{\mathcal{H}\}^{\text{ingroup}}$, the allreduce operation is omitted, as indicated in Algorithm 4.5.

Algorithm 4.5 Chunked Outgroup Sparse Grid Reduce Algorithm

```
procedure ChunkedOutgroupSGReduce($\mathcal{I}^{\text{CT}}, \mathcal{I}^{\text{SG}}$)
    for all $\mathcal{H}_k$ do
        for all $H_{\vec{\ell}'} \in \mathcal{H}_k$ do
            $H_{\vec{\ell}'} = \vec{0}$
        end for
        for all $\vec{\ell} \in \mathcal{I}^{\text{CT}}$ do
            for all $H_{\vec{\ell}'} \in \mathcal{H}_k$ do
                if $C_{\vec{\ell}} \in \mathbf{CG}(H_{\vec{\ell}'})$ then
                    $H_{\vec{\ell}'} \mathrel{+}= c^c_{\vec{\ell}} \cdot C_{\vec{\ell}}[\vec{\ell}']$
                end if
            end for
        end for
        if $\mathcal{H}_k \notin \{\mathcal{H}\}^{\text{ingroup}}$ then
            AllReduce$_{n_g}(\mathcal{H}_k)$    ▷ Reduce hierarchical coefficients across groups
        end if
        for all $\vec{\ell} \in \mathcal{I}^{\text{CT}}$ do
            for all $H_{\vec{\ell}'} \in \mathcal{H}_k$ do
                if $C_{\vec{\ell}} \in \mathbf{CG}(H_{\vec{\ell}'})$ then
                    $C_{\vec{\ell}}[\vec{\ell}'] = H_{\vec{\ell}'}$
                end if
            end for
        end for
    end for
end procedure
```

Sometimes, (some of) the hierarchical coefficients α need to be aggregated on some ranks, for instance for file-based combination (cf. Section 4.5.3), or to analyze the hierarchical surplusses in the context of adaptivity [119]. This is possible by aggregating the data in a separate sparse grid data structure (the 'extra' sparse grid), which of course comes at a certain cost in memory consumption on these MPI ranks.

Conclusively, chunked outgroup sparse grid reduce allows for a parameterizable memory overhead in the whole reduction operation. For this reason, it was used for the experiments in Section 5.4.

4.5. Connecting Two HPC Systems

As mentioned in the previous sections, the 'embarrassing' parallelism of the sparse grid combination technique's PDE solver steps [59] can be used well for process group parallelism, which is added to the traditional parallelism based on domain decomposition. The concept can be taken one step further, as it allows us to distribute the process groups even across different HPC compute systems. This can enable simulations at scales that would otherwise be impossible to perform in the memory of a single HPC system.

One may then call it a *'widely-distributed' sparse grid combination technique simulation*, first introduced in [132], with two machines computing a subset of the combination scheme $\mathcal{I}^{\text{CT}}$ at the same, synchronous, time. Then, the reduction operations outlined in Section 4.4 can be applied in a two-stage process: first to process groups, and then to the different systems. This means that first, a system-local reduction will take place, after which the results are exchanged between the systems. The remote and local sparse grid coefficients are further reduced, and scattered back to the process groups to complete the reduction operation. Due to the CT's multiscale approach, the data volume that needs to be exchanged between systems is always relatively low compared to the number of (full-grid) DOF that are held in memory on the systems. Depending on the size of the combination scheme and the assignment to systems, the SubspaceReduce and OutgroupSGReduce variants (which are the same for the case of two systems) will lead to even lower transfer volumes, to the point where the cost of transfer can become tolerable for synchronous widely-distributed simulations.

The more recent, file-exchange based approach was presented in [138], which along with [132] forms the basis for this section. While the author of this thesis contributed most to the papers, she would like to highlight the contributions by Marcel Hurler and Alexander Van Craen in the design and implementation of the TCP- and UFTP-based protocols, respectively.

It is worth noting that the loose coupling enabled by the sparse grid representation is dramatically cheaper than a traditional domain-decomposition based approach: If all the ghost layer information for a finely-resolved partitioned higher-dimensional simulation had to be communicated across the internet, the exchange would take prohibitively long. The single-system CT alternative—writing all intermediate full grid data to file after every solver update and reading it back in for combination—would also be too slow to be practical.

4.5.1. Added Technical Challenges on Large-Scale Systems

The DisCoTec approach to widely-distributed simulations tries to be as platform-agnostic as possible, and to use the same code base for all systems—even if different MPI implementations, file systems, and libraries are used on the different systems. DisCoTec is started on each HPC system separately, which means that the job start synchronization and remote data connection need to be organized independently of the program. However, vice versa, if the batch job synchronization and data transfer *can* be established for a pair of HPC systems, then it is possible to use the widely-distributed CT with DisCoTec.

In terms of the data transfer, HPC systems are particularly challenging due to high security standards. Typically, the command line access via `ssh` is limited to some specific IP addresses, and may require additional steps such as passwords or two-factor authentication. Often times, the compute nodes themselves will have either a low-bandwidth connection to the outside world (like HAWK), or no connection at all (like SuperMUC-NG). Then, they can only be accessed through the login nodes. For `ssh` tunnels through the login nodes, the systems' firewalls may restrict the direction in which tunnels can be opened and connections can be set up. Fortunately, there are established methods of file transfer between some systems, which avoids talking to the compute nodes directly.

The main challenge, however, remains the joint availability of two HPC systems. For the tests presented in [138] and Section 5.5, queue reservations were used which the HPC centers granted specifically for the experiments. This poses a substantial overhead, but may be worthwhile nonetheless for extreme-scale simulations (which are more likely to be granted during low-occupancy times, such as winter holidays). One can also think of setups with lower organizational effort, such as computing a big part of the combination scenario on a publicly funded machine and complementing the missing part on a commercial cloud system as soon as the larger job starts.

4.5.2. Serial Sparse Grid Data Exchange (Through TCP)

The initial approach to connecting two DisCoTec instances was to use a TCP connection between the two systems. To this end, a chain of `ssh` tunnels needs to be established from a trusted machine (typically with registered IP address) to connect a broker process to the two DisCoTec manager processes [132]. For the widely-distributed reduction part, we select one of the process groups, pg_{tl}, which will be responsible for collecting the local data, sending it to the manager rank,

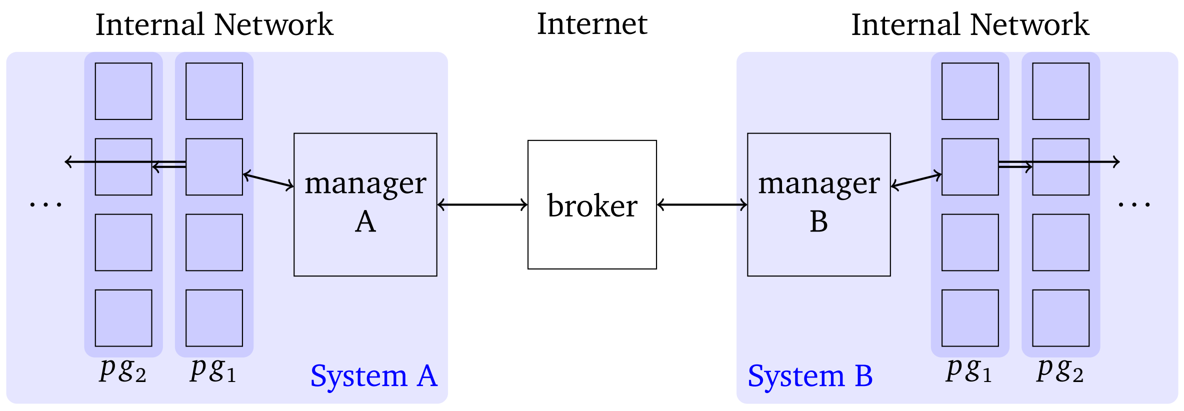


Figure 4.3.: Rank placement, cf. Figure 4.1, and TCP communication set up with two DisCoTec instances distributed over two HPC systems: Each part of the sparse grid is sequentially exchanged both ways through the broker process and the two manager ranks. In this figure, the first group is selected as the communicating group, or $pg_{tl} = pg_1$. Within the systems regular MPI communication is used, and between the systems the data is sent through TCP sockets.

receiving and broadcasting the result back to the process groups. The manager rank on each system will receive the domain-decomposed sparse grid chunk from one of the ranks in pg_{tl} at a time, and either send it to the other system through the broker, or perform the reduction and send the result back to the other system (again, through the broker). The reduced results are then given back to the respective rank of pg_{tl}, and the reduction can continue for the next rank in the process group. Figure 4.3 illustrates the communication pattern for one of the sparse grid chunks. This algorithm ensures that no rank will ever have to hold more than twice the size of a sparse grid chunk in memory.

There are some drawbacks to this approach. Like all connections through the internet, the connection may be of varying quality, since the resources are shared with all other internet users. More specifically, the setup of chained `ssh` tunnels is tedious and results in relatively volatile connections. Also, the data transfer is sequential—the parallelism in the code cannot be used to speed up the transfer of data. Furthermore, wrapping a TCP protocol in an `ssh` tunnel may lead to adverse effects on the transfer speed, as the inner and outer connection will have mismatched parameters [111]. All these factors contribute to transfer speeds below what one would wish for in an HPC setting.

4.5.3. File-Based Sparse Grid Data Exchange (Through UFTP)

Another possible data exchange mode is file transfer. To realize it, it is necessary for DisCoTec to write the sparse grid data to and read it from file in parallel. This is achieved by assigning some *I/O ranks* to aggregate the necessary data and to perform the writing and reading through MPI-IO [113]. Theoretically, the I/O ranks could be one process group, like in the TCP case, but in practice it is more efficient to distribute the I/O ranks across as many nodes as possible, especially if the link from node racks to the file system poses a bottleneck (e. g., on HAWK). For this reason, the I/O ranks $\{r\}_{io}$ are distributed diagonally across the process groups of a DisCoTec instance, i. e.,$r \in \{r\}_{io} iff (r \bmod n_p) \bmod n_g = \lfloor \frac{r}{n_p} \rfloor$, as illustrated by the green ranks in Figure 4.4.

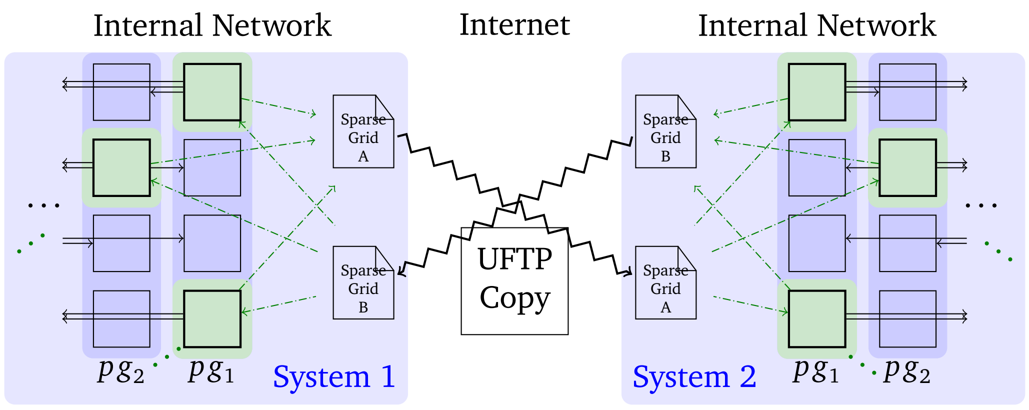


Figure 4.4.: DisCoTec ranks if extended to two systems with the file-based approach, cf. Figure 4.1. On either system, there is a DisCoTec instance with ranks ordered into process groups (pg_i), and green highlights indicate I/O ranks. MPI communication is denoted by solid black arrows, file I/O by dashed green arrows, and the widely-distributed file transfer by jagged black arrows. The UFTP file transfer is handled by separate dedicated processes on gateway nodes. (Figure adapted from [138])

Every I/O rank allocates an extra sparse grid to hold the conjoint data of its spatial domain. In the overall reduce operation with sparse grid reduce, this means that it is sufficient to perform a reduction towards the I/O ranks only, because the entire sparse grid will be updated on the I/O ranks, and broadcast to the other ranks after updating with the other system's data. For ChunkedOutgroupSGReduce, it is necessary to have a horizontal allreduce, such that the sparse grid data not present in the conjoint sparse grid is fully exchanged within the system. The file I/O operations can be performed in parallel—each I/O rank writes out its part of the sparse grid domain as a slice of the file in a collective function call—and the resulting files can be exchanged through separate UFTP processes, illustrated by the jagged black arrows in Figure 4.4. In contrast to the TCP variant, the transfer can take place in parallel if a multi-connection file transfer tool is used, and even overlap the two transfer directions. This is possible since each DisCoTec instance can perform the reduction operation of local and remote sparse grid data on the I/O ranks, and then broadcast the result horizontally to the other ranks.

This idea, as well as the implementation, is entirely independent of the file transfer mode. For [138] as well as the results in Section 5.5, the UFTP tool [149] was used, since it is an established file transfer set up between the systems of the Gauss Center for Supercomputing (GCS), cf. Section 5.1. The parallelization and tuning of UFTP was done by Alexander Van Craen, resulting in transfer speeds of up to 3.8 GB/s between HAWK and SuperMUC-NG [138]. The main drawback of this approach compared to TCP is the additional time needed for writing and reading the sparse grid data to and from file, which can vary significantly between systems, cf. Section 5.5. However, UFTP resolves the issues of parallelization and suitable protocols, and is therefore the preferred method for widely-distributed simulations with DisCoTec on the GCS systems.

4.6. Distributing Combination Schemes for Minimal Data Volume

If a combination scheme is distributed to multiple process groups, a sophisticated distribution algorithm can substantially reduce the amount of data that needs to be communicated. Recall that if $A \subseteq \mathcal{I}^{\mathrm{CT}}$ is assigned to a certain process group, its outward communication volume is given by **sharedG**(A), cf. Equation (4.5). This applies even more when the combination scheme is distributed across different HPC systems. In this case, A denotes the subset of the combination scheme that is assigned to one of the systems. Reducing the costly transfer volume across the network is of utmost importance in this case, as too long transfer times can lead to long waiting times of entire systems. In this widely-distributed setting, we call **sharedG**(A) the *conjoint* sparse grid. For the special case of two HPC systems 1 and 2, the conjoint sparse grid can be written analogously to Equation (4.5) as

$$\mathcal{I}^{\mathrm{cj}} := \{\vec{\ell}' \in \mathbb{N}^d \;\forall s \in \{1,2\} \;\exists\, \vec{\ell} \in \mathcal{I}^{CT,s} : \vec{\ell}' \leq \vec{\ell}\} \;, \tag{4.7}$$

and the communication volume for grids with one-sided (periodic) boundaries is given by

$$\#\mathrm{DOF}^{\mathrm{conjoint}} = \sum_{\vec{\ell}' \in \mathcal{I}^{\mathrm{cj}}} 2^{\left|\vec{\ell}'\right|_1} \,. \tag{4.8}$$

The author of this thesis developed two different approaches to distributing the scheme, which were published in [138]: A greedy, but less flexible approach, and a more flexible, but more expensive heuristic approach. Both assign the grids on the *main diagonal* of the combination scheme, cf. Section 2.3, to either system; The lower diagonals are then assigned in a fashion that does not increase $\#\mathrm{DOF}^{\mathrm{conjoint}}$. While the two methods were developed for the widely-distributed setting, they also apply to distribution across multiple process groups on the same system, within the specified limitations. The implementations are openly available at [128].

4.6.1. Symmetric Splitting by the Level-Sum Criterion

The level-sum criterion applies to symmetric splits, i. e., where the number of dimensions is a multiple of the number of systems, and each system is assigned the same share of work. In particular, this means that the number of scheme partitions cannot be higher than the number of dimensions. Then, each system can be assigned preferred dimensions $j \in \{1, \dots, d\}$. One may assign system one to the odd and system two to the even dimensions through a preference list p_s: $p_1 = 1, 3, \dots, p_2 = 2, 4, \dots$. The mixed level vectors belonging to the systems

$$\ell_j^{\mathrm{mix}} = \begin{cases} \ell_j^{\max} & \text{if} \quad j \in p_s \\ \ell_j^{\min} & \text{else} \end{cases} \tag{4.9}$$

are corners of the hypercube in the level space that contains the index set $\mathcal{I}^{\mathrm{CT}}$.

One may imagine the process as ‘starting’ at the corners of the main diagonal of the combination scheme index set $\mathcal{I}^{\mathrm{CT}}$, which is a simplex. Each corner’s level vector is assigned to the system that ‘prefers’ the dimension $\arg\max_j \ell_j$ in which the corner is located. Assuming a large combination scheme, the corner’s neighboring elements on the main diagonal are then cheapest assigned to the same system as the corner itself. Iterating this procedure, one can derive a greedy assignment of the entire main diagonal of the combination scheme, which is termed the *level-sum criterion*:

$$s(\vec{\ell}) = \begin{cases} 1 & \text{if} \left|\vec{\ell} - \vec{\ell}^{\text{mix},1}\right|_1 < \left|\vec{\ell} - \vec{\ell}^{\text{mix},2}\right|_1 \\ 2 & \text{if} \left|\vec{\ell} - \vec{\ell}^{\text{mix},1}\right|_1 > \left|\vec{\ell} - \vec{\ell}^{\text{mix},2}\right|_1 \end{cases} . \tag{4.10}$$

In those cases where the norms $\left|\vec{\ell} - \vec{\ell}^{\text{mix},s}\right|_1$ are equal, either system will be assigned in a round-robin fashion. This ambiguity, i. e., the number of grids where the norms in Equation (4.10) are equal, can be reduced if a value slightly smaller than 1 is used for the vector norm; The current implementation uses $|\cdot|_{0.99}$.

In fact, Equation (4.10) can also be applied to the lower diagonals of the combination scheme. This greedy assignment is deterministic and can be computed quickly enough to be used in-situ during the program's run time on an HPC system, but it is restricted to the symmetric case, which is often not sufficient as we are going to see in Section 5.5.

4.6.2. Heuristic Splitting with the METIS Graph Partitioner

Attempts to extend the level-sum criterion to arbitrary dimensionalities and varying shares of work resulted in suboptimal assignments. Thus, the second approach is designed to provide this flexibility. It uses the heuristic graph-partitioning algorithm METIS [90] to assign the main diagonal to the systems.

The graph is built up with the levels $\vec{\ell}$ on the main diagonal as nodes. Edges between nodes are introduced if the corresponding levels are neighbors in the main diagonal simplex. All levels on the simplex have the same level sum, which means that $\vec{\ell}_i$ and $\vec{\ell}_j$ are neighbors iff $\left|\vec{\ell}_i - \vec{\ell}_j\right|_1 = 2$. Accordingly, the possible number of neighbors is bounded by $\binom{d}{2}$ per node. The number of parts and their respective relative sizes are free parameters of the METIS algorithm. The communication-volume based METIS multi-level graph partitioning [90] is applied to the graph, and the returned partitions are required to be contiguous. Then, the lower diagonals can, again, be assigned in a way that does not increase the communication volume.

Unfortunately, the run time of the METIS algorithm is not known in advance and may become excessively long, depending on the specified parameters. Accordingly, the grid assignments (and the pertaining sparse grid sizes) have to be computed as part of the combination scheme preprocessing. But this heuristic approach appropriately deals with two important use cases: First, the combination scheme can be distributed between two different systems, even if one of the systems should get a much larger share of work. And second, it can be used to distribute the combination scheme between multiple process groups on the same system, because the number of process groups typically exceeds the number of dimensions. Conclusively, this algorithm was also used to optimize the process group assignment for the ChunkedOutgroupSGReduce algorithm in Sections 5.4.2 and 5.5.

4.7. New Contributions to DisCoTec

In addition to the algorithms and optimizations described above, the author of this thesis contributed to other aspects of the DisCoTec framework[1]. The following list of contributions is not exhaustive, but showcases the different areas in which the code has improved over the course of the presented work.

Refined Numerics

- Monte-Carlo error measurement functions, based on point-wise interpolation at arbitrary coordinates
- ghost layer and boundary exchange functions on `DistributedFullGrid` (necessary to implement the advection solver from Section 3.4 directly in DisCoTec)
- Discontinuous Galerkin-compatible full grid implementation: `DFGEnsemble`
- mass-conserving basis transformations / hierarchization and dehierarchization according to Section 2.1.1 (joint work with Katharina Kormann and Johannes Rentrop)
- minimum level in basis transformations according to [137]
- Kahan summation within process group in LocalReduce operation (main driver for increased accuracy from [136] to [137])

[1]Most of the development progress is documented as pull requests at
`https://github.com/SGpp/DisCoTec/pulls?q=`

Solvers in the Framework

- further developed and performance-engineered advection FDM/FVM solver, cf. Sections 3.4 and 5.4 (joint work with Alexander Van Craen and Marcel Breyer)
- DisCoTec interface for hyper.deal (a deal.ii based Vlasov code [114]) (joint work with Peter Münch and Marius Göhring)
- DisCoTec interface for SeLaLib, cf. Section 3.5 (joint work with Katharina Kormann and Michael Obersteiner)

Combination Technique Features

- dynamic rescheduling of tasks during run time, for solvers that change in load (joint work with Marvin Dostal [36])
- allow reading combination schemes from `.json` file and the (conjoint) sparse grid sizes from binary `.sizes` files, along with the corresponding tools to generate these files: combischeme utilities [128] and DisCoTec's `subspace_writer`.
- 'worker-only' mode without manager rank, cf. Figure 4.1, with static grid assignment to process groups: either round-robin, statically load balanced based on the number of DOF, or from file
- combination / reduction variants SUBSPACEREDUCE, OUTGROUPSGREDUCE, CHUNKEDOUTGROUPSGREDUCE, cf. Algorithms 4.3 to 4.5
- write and read sparse grid data as binary files with MPI-IO [113], and file-based combination algorithm based on this feature
- widely-distributed combination protocols (joint work with Marcel Hurler for the TCP variant [132] and with Alexander Van Craen for the UFTP variant [138])

Savings in Allocated Memory

- avoid allocations of communicators and temporary data structures
- shared-memory parallelism (OpenMP), which allows to have less memory overhead in communicators and data structures per node

- memory overhead measurement for `DistributedFullGrid` as part of continuous integration tests
- optimized `DistributedFullGrid` data structures to avoid growing memory overhead with growing n_p
- saving memory in data copies between the `Task`, `DistributedFullGrid`, and `DistributedSparseGrid` structures (partly joint work with Marcel Hurler)
- one-sided boundary for `DistributedFullGrid`, which is natural for periodic boundary conditions, and allows for better grid point load balancing

Chapter 5

Scaling DisCoTec Up to Full HPC Systems and Beyond

This chapter presents and discusses run time measurements for DisCoTec on different HPC systems.

The first set of measurements, in Section 5.3.1, presents the run time measurements of simulations with DisCoTec and the gyrokinetic Vlasov solver GENE [56, 89, 160] on HAWK and SuperMUC-NG as part of the EXAHD project, in which the presented research originated. While the measurements were already taken in 2021, with now-outdated versions of the codes, they motivated much of the further progress in DisCoTec and are therefore presented here. Also, the measurements allowed to discover the mismatched memory expectations between DisCoTec and GENE at scale. Additionally, the different scaling setups in the measurements helped to refine the general methodology for scaling measurements with different combination schemes as described in Section 5.2.

The second set of measurements, in Section 5.4, presents the results of scaling advection simulations with recent versions of DisCoTec up to full system sizes on four different HPC systems—Fritz, HAWK, JUWELS Cluster, SuperMUC-NG—which are introduced in Section 5.1. These measurements contain both a strong scaling scenario with a very low memory footprint (less than 40 GiB of component grid data to be distributed among thousands of cores), and a weak scaling scenario. The weak scaling scenario aims to approach the memory limits of the considered systems with a memory footprint of more than 1.1 GB of component grid data per core. The measurements show that it is possible to scale DisCoTec up to full system sizes on all considered systems, and use a substantial amount of memory for the higher-dimensional solver while doing so.

The third set of measurements, in Section 5.5, evaluate the simulation performance of a widely-distributed scenario on SuperMUC-NG and JUWELS, and compare them to a recently published setup connecting the SuperMUC-NG and HAWK systems. The target scenario is so large that it cannot be placed on SuperMUC-NG alone, but requires the combination of SuperMUC-NG and JUWELS. The measurements present the run times measured for $\frac{1}{15}$ of the scenario, and discusses challenges encountered at this scale. It gives an outlook on potential solutions to performing the target simulation, which itself is left for future work.

All experiments consider the hierarchical hat basis functions. While for the GENE simulations, the mass-conserving basis functions had not yet been sufficiently researched, for the advection simulations, the hierarchical hat basis functions were chosen because they provide the most efficient and well optimized transformations [74, section 3.3] in DisCoTec. Also, Chapter 3 showed that the hierarchical hat functions lead to stable and relatively accurate solutions for the considered advection scenario well up to hundreds of advection solver timesteps per combination step.

5.1. Comparison of HPC Systems Used for Experiments

DisCoTec was benchmarked on all three CPU-based Tier-0 / Tier-1 HPC systems currently managed through the Gauss Centre for Supercomputing (GCS) [81], which are JUWELS Cluster [27] at the Jülich Supercomputing Center (JSC), HAWK [80] at the High Performance Computing Center Stuttgart (HLRS), and SuperMUC-NG [72] at the Leibniz Supercomputing Centre (LRZ). ('JUWELS Cluster' will subsequently be called only 'JUWELS'). Furthermore, DisCoTec was tested on the Tier-2 / Tier-3 HPC system Fritz [50], which is operated by the Erlangen National High Performance Computing Center (NHR@FAU).

Although GPGPUs are becoming increasingly important for scientific computing, the current TOP500 list [153] of HPC systems still contains 315 CPU-only systems, without accelerators. Of these, 258 are based on Intel Xeon processors and 46 on AMD EPYC processors. Thus, by looking at results on Fritz, JUWELS, and SuperMUC-NG with different Intel Xeon processor generations, and Hawk with its AMD EPYC Rome CPUs, the measurements should still be fairly representative of the current HPC landscape.

	Fritz [50]	HAWK [80, 82]	JUWELS Cluster [27]	SuperMUC-NG [72]
# nodes	992	5632	2271	6336
# core/node	72	128	48	48
# cores total	71,424	722,176	109,008	304,128
base frequency	2.4 GHz	2.25 GHz	2.7 GHz	2.3 GHz (w/o EAR [42])
main memory per core	3.56 GiB/core	2 GiB/core	2 GiB/core	2 GiB/core
main memory spec. (channels per node)	DDR4, 16 × 3200 MHz	DDR4, ?? × 3200 MHz [73]	DDR4, 12 × 2666 MHz	DDR4, 12 × 2666 MHz
(assumed) memory bandwidth per node	409.6 GiB/s	380 GiB/s [34]	255.9 GiB/s	256 GiB/s
processor model	Intel Xeon Platinum 8360Y 'Ice Lake'	AMD EPYC 7742 'Rome'	Intel Xeon Platinum 8168	Intel Xeon Platinum 8174 'Skylake'
(highest) vectorization	AVX512	AVX2 [34]	AVX512	AVX512
interconnect	Blocking HDR100 Infiniband	InfiniBand HDR	InfiniBand EDR (Connect-X4)	Intel Omni-Path
file system	Lustre	Lustre	GPFS	GPFS
TOP500 position [153]	178	32	102	31
MPI version used for Section 5.4	Intel MPI 2021.7	MPT 2.26	OpenMPI 4.1.4	Intel MPI 2019.12
`MPI_Allreduce` implementation used	binomial gather + scatter	2 (not documented)	nonoverlapping	binomial gather + scatter

Table 5.1.: Hardware properties of the 'standard' nodes of the machines used, in alphabetical order.

The parallelism and hardware terminology used in the chapter is going to be as follows: *Processes* are the same as MPI *ranks*, where each DisCoTec instance has either $n_g \cdot n_p$ or $n_g \cdot n_p + 1$ processes, depending on whether a manager rank is used or not. n_g denotes the number and n_p the size of the groups. For Section 5.3, this is going to be the same as *cores*, since no OpenMP threading was used. For the advection simulations, however, *cores* will be equivalent to OpenMP *threads*. Then, the total number of cores used by DisCoTec can be computed as $n_g \cdot n_p \cdot n_t$ (without manager rank), where the number of threads per rank n_t was uniformly set to 4 throughout Sections 5.4 and 5.5.

5.2. Challenges for Scaling Up DisCoTec

Section 5.3's extensive tests with DisCoTec and GENE, also in the light of main memory consumption, helped to develop a simulation setup that allows to scale DisCoTec experiments up to full HPC system sizes in a well-interpretable manner. As mentioned previously, there are two ways of scaling up DisCoTec, either with more or larger process groups. Larger groups (expanding vertically in Figure 4.1) correspond to a higher level of 'traditional' parallelism based on domain decomposition, where slices of the data need to be communicated to the Cartesian neighbors (or along Cartesian poles). Conversely, more groups (expanding horizontally in Figure 4.1) add the parallelism specific to the CT, which is embarrassingly parallel for the solver update and basis transformation steps. (See also the arrows in Algorithm 4.1 for the communication directions.) Both types of parallelism are necessary to effectively scale up to full HPC system sizes. When scaling up, it is important to carefully construct the communication volume, since some aspects of the CT can be counter-intuitive.

Strong Scaling

For a strong scaling scenario, the amount of work, in our case the combination scheme defined by $\mathcal{I}^{\text{CT}}$, stays exactly the same. It is going to be distributed among more and more processes, either larger or more process groups. The time to solution for p processes is denoted by T_p. One can then define the speedup as

$$S(p) := \frac{T_1}{T_p}. \tag{5.1}$$

Ideally, the speedup would be equal to the number of processes, which of course will not be the case due to serial and parallel overheads. For every program, the communication will at some point dominate the computation (unless it is trivially parallelizable), but the question is when that happens. The success of strong scaling can be measured by the strong scaling efficiency, which is defined as the ratio of the ideal speedup and the actual speedup, i. e.,

$$\epsilon_s(p) := \frac{S_p}{p} = \frac{T_1}{pT_p}. \tag{5.2}$$

The ideal efficiency would be close to 1, but the value is bound to decrease for larger numbers of processes. For the combination step in particular, adding more and more groups can lead to the dominance of the $\mathcal{O}(\log n_g)$-complexity allreduce operation. Depending on the problem size and reduction algorithm, it may also become necessary to perform more intermediate allreduce operations per step as the number of groups increases. On the other hand, enlarging the groups will lead to a dominance of full-grid data communication over the solver and basis transform computation.

Weak Scaling

For the weak scaling scenario, the amount of work, in our case the combination scheme, is increased proportionally to the number of processes. This in turn can be achieved in two different ways: Either the resolution levels of the grids are increased, or more grids are added. The experiments in Section 5.4.2 use increased resolution levels for larger groups, and added grids for added groups. This is beneficial for run time analyses, since the number of assigned component grids and their number of DOF per process will remain approximately constant, which best fits the spirit of a weak scaling setup. If the Cartesian processes in the process groups are added in the same direction in which one refines the combination scheme, then also the anisotropic resolutions are kept constant.

The—numerically more interesting—approach of increasing the distance between $\vec{\ell}^{\min}$ and $\vec{\ell}^{\max}$ in the truncated combination scheme was used for the experiments in Section 3.4. However, it is not considered in the timing measurements, since it changes a range of parameters at once, which makes it very hard to distinguish different effects on the run time.

Ideally, if everything was perfectly parallelizable, the timing of the simulation would stay approximately the same. Now, the success of weak scaling can be measured with the weak scaling efficiency

$$\epsilon_w(p) := \frac{T_1}{T_p} \tag{5.3}$$

Again, the ideal efficiency would be close to 1, but the value is bound to decrease as the process count goes up.

Load Imbalance

Which type of parallelization works better—larger or more process groups—will depend on the assignment of grids to groups. Since the aim of this work is to approach the memory limits of full HPC systems, grid assignments are mainly based on the number of degrees of freedom. This means that large process groups which are assigned a (relatively) high number of (relatively) coarse/small grids will have a higher (relative) communication overhead, and thus display a lower parallel efficiency in the process group size n_p. The effect leads to a divergence of memory footprint and run time for the different process groups. The experiments in Section 5.4 use ChunkedOutgroupSGReduce, which means that the assignment of grids to groups will also influence the allreduce's communication volume: For a good assignment (with much ingroup data), the communication volume will be very low for two process groups and increase with the number of groups. The reduce operation's division into chunks will then lead to discrete increases in the communication times. For the more careless approach of assigning the grids to groups only based on the number of DOF and not on neighborhood in $\mathcal{I}^{\text{CT}}$, which is used for the strong scaling setup in Section 5.4.1, the communication volume will be relatively high for two process groups already, and remain on a similar level as process groups are added.

This work uses the same definition as [79] for the load imbalance L: Load imbalance is the maximum load of any rank divided by the average load of all ranks, or

$$L := \frac{\max\limits_{r\in\{0,\dots,(n_g\cdot n_p-1)\}} T_r}{\frac{1}{n_g\cdot n_p}\sum\limits_{r\in\{0,\dots,(n_g\cdot n_p-1)\}} T_r}. \tag{5.4}$$

Here, T_r, the time spent in the solver update step for rank r, serves as a proxy for load on the rank. The ideal value for the load imbalance would, again, be 1.

Load imbalance poses a problem for both strong and weak scaling. The main effect was hinted at in Figure 4.2: Run time differences between process groups will lead to idle time for the faster groups, which is wasted time for those. In addition, across the whole MPI program, the typical effects of OS jitter and random variations in the node hardware will lead to load imbalances. (These will typically be more severe within the process group, since more implicit synchronization happens in the process groups.)

With DisCoTec, one can influence the load imbalance *between* the process groups by component grid assignments to groups, but the load imbalance *within* the process groups can only be influenced to a limited extent, since all component grids have to share the exact same domain decomposition.

5.3. DisCoTec And The Gyrokinetic Solver GENE

The aim of the EXAHD project in which this research originated was the application of the sparse grid combination technique to the gyrokinetic Vlasov solver GENE for time-stepping simulations.

GENE [160] is a plasma turbulence solver that is used to model state-of-the-art fusion experiments [52, 89, 168]. An important feature of GENE is that it uses the gyrokinetic approximation [14], which effectively reduces the number of physical space dimensions from six to five. The collisionless Vlasov Equation then reads

$$\frac{\partial F}{\partial t} + \nabla_{\vec{x}} F \cdot \frac{\mathrm{d}\vec{x}}{\mathrm{d}t} + \frac{\partial F}{\partial v_{\|}} \cdot \frac{\mathrm{d}v_{\|}}{\mathrm{d}t} + \frac{\partial F}{\partial \mu} \cdot \frac{\mathrm{d}\mu}{\mathrm{d}t} = 0\,, \tag{5.5}$$

where $v_{\|}$ and μ are new, transformed velocity variables. GENE employs a fourth-order finite difference scheme to solve the Vlasov–Maxwell equations in the five-dimensional phase space, and also fourth-order explicit time stepping. Special treatment is applied to the x_2 direction, where the solution is represented in Fourier space. If high velocities occur, e. g., due to high electric field gradients, the time step is adaptively shortened. For use with DisCoTec, this means that the combination interval is set as a parameter in time units, but in each combination step, any of the full grid solver instances may need different amounts of time steps to reach the given combination interval. Furthermore, each of the dimensions needs a minimum resolution per process, which is why the truncated CT (cf. Section 2.3) was used for the experiments. GENE uses complex double precision floating point numbers to represent f, which can be interesting in terms of the memory footprint of the simulations ($\Rightarrow$ 16 B per DOF).

This approach had shown promising first results for linear, local simulations [100], but the nonlinear simulations had three challenges remaining, namely the load balancing, the memory footprint and the stability of the simulation. As mentioned previously in Section 3.1, the simulation would become unstable for longer simulations (and this could potentially be addressed with the mass-conserving basis functions as discussed in Chapter 3). This section focuses on the more technical aspects, which are related to scalability and memory consumption of GENE+DisCoTec simulations.

5.3.1. Scaling DisCoTec with GENE

The goal of these experiments is to extend the existing run-time analysis with DisCoTec to adiabatic electron non-linear global [89] GENE simulations (i. e., those that use one kind of particle, display turbulent plasma movement, and model the whole tokamak torus), while comparing the two systems HAWK and SuperMUC-NG, cf. Table 5.1, for their run time behavior. By contrast, Heene [74] compared CT run times with the linear GENE solver based on run time estimates [74, Figure 3.26] or on relatively small scenarios [74, Figure 5.17].

The measurements were taken in November 2021 with commit `57dceaa2` of DisCoTec and commit `d19569eb` of GENE[1]. For this reason, some features were not yet developed for DisCoTec, such that the experiments were performed with an additional manager rank (which indeed makes sense for dynamic load balancing [135]), with boundary points on all sides, and with the sparse grid reduce algorithm, cf. Algorithm 4.2, but without both OpenMP shared-memory parallelism and minimum hierarchization levels. Combination is performed after every 0.1 GENE time units.

Three setups are considered: 'More Process Groups for the Same Problem' (i. e., strong scaling in n_g), 'Larger Process Groups for Larger Problems' (i. e., weak scaling in n_p), and 'More Process Groups for Larger Problems' (i. e., weak scaling in n_g). Together, they help understand why scaling up GENE and DisCoTec jointly can be challenging.

More Process Groups for the Same Problem

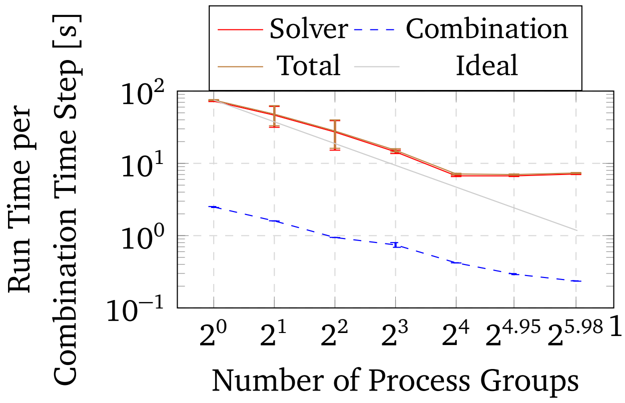
The strong scaling in the number of process groups n_g—horizontally in Figure 4.1—uses the following combination scheme for groups of 512 processes ($n_p = 512$).

Combination scheme for GENE strong scaling, n_g groups of 512

$\vec{\ell}^{\min}$	$(4, 5, 3, 4, 2)$	# grids	121
$\vec{\ell}^{\max}$	$(9, 5, 8, 9, 7)$	# finest grids	56
total FG #DOF	6.69×10^8	mem. finest grids	128 MiB
total FG memory	9.97 GiB	#DOF FG at $\vec{\ell}^{\max}$	2.75×10^{11}

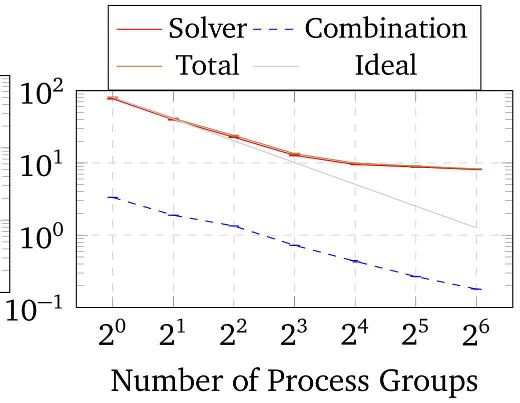
[1] to be found in fork
`https://gitlab.mpcdf.mpg.de/g-michaelobersteiner/gene-dev.git`

The five dimensions denote $x_1, x_2, x_3, v_{\|}, \mu$ (or (`x, y, z, v, w`) in GENE's internal notation), respectively. x_1 denotes the radial direction, for which fine resolutions are particularly interesting in GENE—and unfortunately, also relatively expensive in run time. The CT is applied only to the $x_1, x_3, v_{\|}, \mu$ directions, as the Fourier-transformed x_2 direction has its own pitfalls [99].



(a) Strong scaling on HAWK

Due to HAWK's topology-aware scheduling, higher node counts cannot be freely chosen, resulting in slightly-less-than-power-of-two worker counts to accomodate the manager rank.



(b) Strong scaling on SuperMUC-NG

Figure 5.1.: Strong scaling timings for process groups of size 512: Times are shown as means over all time steps and worker processes, the error bars denote the standard deviations. Since the total time is dominated by the solver time, the lines almost overlap.

Looking at the results in Figure 5.1, it appears that the solver time step does not get any faster for more than 16 process groups. This can be explained by the run time imbalance between the tasks—up to $n_g = 16$ groups, there were enough component tasks to distribute such that the load could be balanced between process groups. But once the one instance that operates on a grid with very fine x_1 resolution takes longer than the other instances together, divided by the number of remaining process groups, all other process groups have to wait for that instance to finish [135].

For this particular scenario, the run time imbalance between the tasks puts a virtual end to scalability for more than 16 process groups. But prior to that, adding process groups scales nicely, and the behavior between the two machines is very similar.

Larger Process Groups for Larger Problems

Performing weak scaling requires the design of a set of combination schemes to be distributed among more and more processes. To this end, the smallest combination scheme is distributed among 64 processes with a parallelization vector $p = (2, 1, 2, 4, 4)$, i. e., the number of MPI processes in each direction. x_1 is the direction in which $\vec{\ell}^{\min}$ and $\vec{\ell}^{\max}$ as well as p are increased first: We then iterate the combination directions starting from x_1, increasing the minimum and maximum level and doubling the number of ranks in that direction:

Combination scheme for GENE weak scaling, one group of $n_p = 64$

$\vec{\ell}^{\min}$	$(3, 5, 2, 3, 2)$	# grids	121
$\vec{\ell}^{\max}$	$(8, 5, 7, 8, 7)$	# finest grids	56
total FG #DOF	8.36×10^7	mem. finest grids	16 MiB
total FG memory	1.25 GiB	#DOF FG at $\vec{\ell}^{\max}$	3.44×10^{10}

. . .

Combination scheme for GENE weak scaling, one group of $n_p = 512$

$\vec{\ell}^{\min}$	$(4, 5, 3, 4, 2)$	# grids	121
$\vec{\ell}^{\max}$	$(9, 5, 8, 9, 7)$	# finest grids	56
total FG #DOF	6.69×10^8	mem. finest grids	128 MiB
total FG memory	9.97 GiB	#DOF FG at $\vec{\ell}^{\max}$	2.75×10^{11}

Combination scheme for GENE weak scaling, one group of $n_p = 1024$

$\vec{\ell}^{\min}$	$(4, 5, 3, 4, 3)$	# grids	121
$\vec{\ell}^{\max}$	$(9, 5, 8, 9, 8)$	# finest grids	56
total FG #DOF	1.34×10^9	mem. finest grids	256 MiB
total FG memory	19.94 GiB	#DOF FG at $\vec{\ell}^{\max}$	5.50×10^{11}

. . .

Combination scheme for GENE weak scaling, one group of $n_p = 8192$

$\vec{\ell}^{\min}$	$(5, 5, 4, 5, 3)$	# grids	121
$\vec{\ell}^{\max}$	$(10, 5, 9, 10, 8)$	# finest grids	56
total FG #DOF	1.07×10^{10}	mem. finest grids	2048 MiB
total FG memory	4.98 TiB	#DOF FG at $\vec{\ell}^{\max}$	4.12×10^{12}

Note that this set contains the scenario used for the strong scaling setup above.

The next larger set of scenarios (with $\left|\vec{\ell}^{\min}\right|_1$ and $\left|\vec{\ell}^{\max}\right|_1$ higher by one) would not have fit onto the main memory of the process group for the whole range of parallelizations; The memory requirements listed above are based on the calculated size for storing f and not on the entire GENE memory footprint. In fact, for GENE one has to take special care as the memory and run time overhead increases superlinearly (approximately quadratically) depending on x_1 resolution. This is mainly attributed to the so-called gyro matrix in the gyrokinetic solver. In order to still deal with a fair weak scaling scenario, the methodology mentioned in [52] was employed: The described approach increases the x_1 'box size' of the GENE simulation domain along with the number of points, such that the spatial resolution in x_1 direction is kept constant. This results in a gyro matrix of approximately the same size per component task as the scenario is scaled up (but it will have different sizes within the component tasks of a single scenario).

The combination interval is again 0.1, and from GENE's solver output, it can be observed that for each scenario the number of solver time steps varies between grids and even combination time steps, yet it is similarly distributed among the different runs.

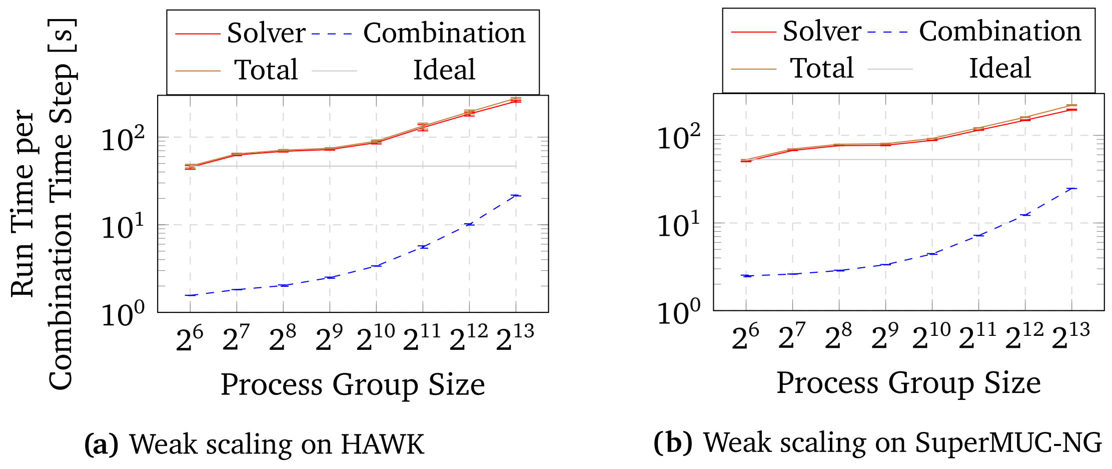
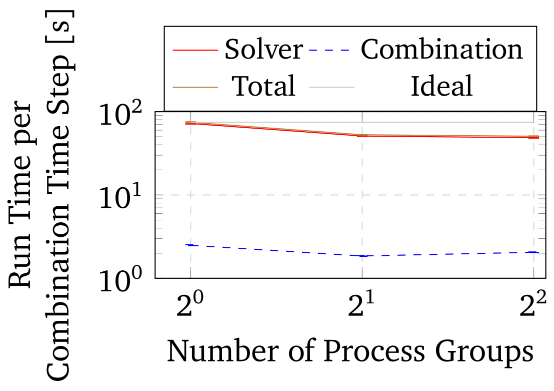


Figure 5.2.: Weak scaling timings with a single process group: Times are shown as means over all time steps and worker processes, the error bars denote the standard deviations. As each level index ℓ_j is increased in round-robin fashion, the parallelization vector p_j in that dimension is doubled.

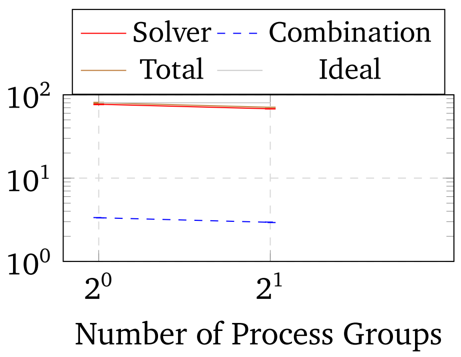
Figure 5.2 shows that the domain-decomposition approach to weak scaling does not work particularly well towards higher process group sizes. The similarity between Figure 5.2b and Figure 5.2a hints that the run time increase for higher degrees of parallelism is not machine-dependent.

There are several reasons to this: Each of the 121 individual GENE tasks has only few points in most of the directions—contrary to most of GENE's optimizations which assume good exploitation of the available memory for each monolithic solver instance. It may even occur that there are so few points per process that it is necessary for GENE not only to communicate to the direct Cartesian neighbors in the MPI communicator, but even to the next neighbors. The same holds true for the basis transformations in the combination step, especially the dehierarchization: Since no minimum level was used for the hybrid representation of f, cf. Equation (2.25), the hierarchization in dimension j will have to communicate with most of the Cartesian neighbors in j direction. Profiling with the help of Martin Ohlerich at LRZ provided the additional insight that the power-of-two domain decompositions, together with boundary points on both sides, leads to significant load imbalances *within* the process groups. The effect is due to the component grids having to have a resolution of $2^{\vec{\ell}} + \vec{1}$, so the 'middle point' always is an additional burden. This insight motivated the adaptation of DisCoTec to true periodic boundary conditions, cf. Section 4.7.

More Process Groups for Larger Problems: Severely Limited



(a) Weak scaling on HAWK



(b) Weak scaling on SuperMUC-NG

Figure 5.3.: Weak scaling timings when adding process groups: Times are shown as means over all time steps and worker processes, the error bars denote the standard deviations. The process group size is 512, and the scenarios are the same as for the same total number of processes in Figure 5.2.

Considering the weak and strong scaling measurements, it may appear sensible to perform weak scaling experiments (which operate on the memory limit) with process-group number scaling (because it scales well as long as the run time is not dominated by one task). To this end, we employ the same scenarios as above for groups of $n_p = 512$ and more, but distribute them among increasing numbers of groups with $n_p = 512$, instead of larger groups.

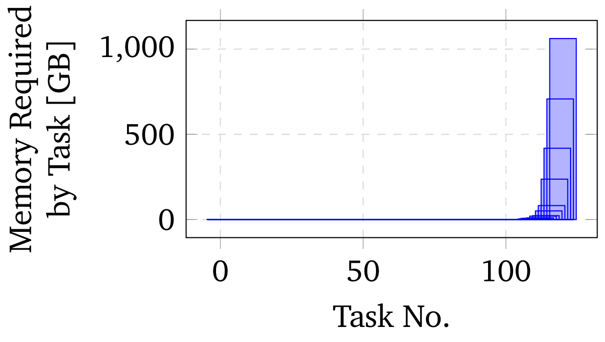
Figure 5.3 shows that, indeed, this leads to excellent scaling; Essentially, no increase in run time can be observed. But the experiments failed for more than two process groups on SuperMUC-NG and more than four process groups on HAWK. The reason lies in the measured memory footprints for the different tasks displayed in Figure 5.4a: GENE's memory scaling, which is quadratic in the x_1 resolution, leads to an enormous memory imbalance *between* the tasks for higher numbers of points in x_1 direction.

For the scenario which can easily be run on one process group of $n_p = 4096$, there is no possible assignment to eight process groups of 512, although this maps to the same amount of resources used. The largest of the scenario's tasks requires 1061.9 GB of main memory, while on the machines tested, there are only 2 GB available per core, and each process group would have 1024 GB available at maximum. The sketch in Figure 5.4b illustrates why this is going to be even more of a problem when going to finer and finer resolutions, i. e., to higher $\ell_{\mathrm{x}}^{\max}$ values: The anisotropy will become more and more dominant.

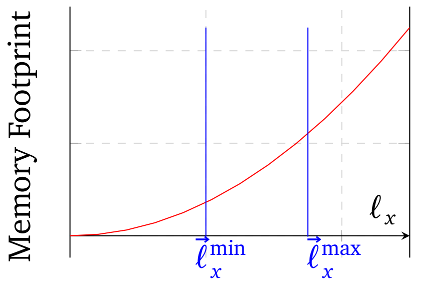
5.3.2. GENE and DisCoTec: The Right Setup for Exascale?

The series of experiments detailed in Section 5.3.1 can lead us to two possible conclusions: First, the optimal parallelization strategy for a given GENE CT scenario can be obtained by choosing the process group size as small as possible (to still fit the task with the largest memory footprint), and then adding more process groups until load imbalance arises between process groups (and the run time is dominated by the most 'expensive' component grid). This approach is practical as long as the combination time is negligible compared to the solver update steps, which was the case for these measurements.

Second, the algorithmic approach of having a uniform domain decomposition on all component grids in DisCoTec is not optimal for use with GENE. The most expensive grids in run time and memory consumption—which will typically be the same ones—had better be distributed among more processes, and the 'smaller' grids should be distributed among fewer processes. However, the ramifications of



(a) Measured: Memory of component tasks in the $\vec{\ell}^{\min} = (5,5,4,4,3), \vec{\ell}^{\max} = (10,5,9,9,8)$ scenario on one process group of 4096 on HAWK. Due to the imbalance in memory—depending on ℓ_x of every component grid—the scenario cannot be run on groups of 512. The largest task consumes 1061.9 GB, and the systems tested only have 2 GB per core.



(b) Model: Memory of component tasks over the x resolution level if the dominating x direction has quadratic scaling. For a typical combination scheme, there will be many grids with $\ell^{\min}_{\mathrm{x}}$ and just a single one with $\ell^{\max}_{\mathrm{x}}$.

Figure 5.4.: Measured and modeled memory requirements for GENE component tasks.

non-uniform domain decomposition have been explored in detail in [112], and it was shown that non-uniform decomposition for DisCoTec can lead to even more problems on the memory limit, when ranks need to collect data from a larger domain than they can store.

Knowing the results from Section 3.5, it is well possible that the mass-conserving basis functions could be used to introduce stability to CT simulations with GENE. But even if that were the case, GENE and DisCoTec would still not be the perfect match at exascale, since through the mismatched memory requirements, it is impossible to fulfill this chapter's promise of 'scaling up to full HPC systems'.

Conclusively, one can note that Vlasov solvers at fine resolution levels (and therefore, high processor counts) are best used with DisCoTec when their memory requirements scale approximately linearly with the number of DOF in the distribution function f, such as is the case with SeLaLib.

5.4. DisCoTec with the Advection Solver: Comparing Four Systems

In contrast to the last section, the experiments in this section are performed with the advection solver as introduced in Section 3.4, which takes some of the algorithmic complexity of Vlasov simulations away, yet still serves as a suitable proxy for Vlasov solvers in many aspects. In particular, typical Vlasov solvers will be bandwidth-limited on most machines, since the structure of the PDE makes it very similar to an advection problem (or rather, two coupled advection steps, which for instance the Cheng–Knorr splitting makes use of [21]). At the same time, the advection solver is significantly easier to performance-model than Vlasov solvers, as it treats all dimensions (numerically) equal.

The measurements in this section were performed in July-Sept 2023 with fairly recent versions of DisCoTec, and thus makes use of the new algorithms and improvements described in Section 4.7. This includes the support for OpenMP-threaded applications, where each DisCoTec rank / process uses four threads to parallelize the computation in shared memory. The uniform setting of four threads was chosen because it is a denominator of the number of cores per node on all machines considered, cf. Table 5.1, and also because of the particular architecture of the AMD Zen2 architecture chip on HAWK: Even within a socket, the access to the L3 cache is non-uniform, and only four cores share the same (relatively small, 16 MB) L3 cache [34] (they form a 'core complex' in AMD's terminology).

All the measurements use ChunkedOutgroupSGReduce, where the strong scaling was measured entirely without I/O and the weak scaling was performed with intermediate write/read operations for (conjoint) file-based combination. Significant differences were found for different implementations of `MPI_Allreduce` of the respective MPI versions. The implementations used on the different systems are listed in Table 5.1; They were selected by the lowest timings attained for the ($n_g = 64, n_p = 1$) strong scaling scenario on each system. For the basis transformations, the minimum hierarchization level was set to the minimum level of the respective combination scheme, and no manager rank was used. The exact commit hashes and data repositories are summarized in Section 6.2 as *Reproducibility References*.

5.4.1. Strong Scaling with a Small Combination Scheme

For the strong scaling case, some amount of 'work', i. e., a combination scheme defined by $\mathcal{I}^{\text{CT}}$, is distributed among increasing numbers of processes. We are particularly interested in the parallel efficiency and the balance between the solver and combination times for the different parallelization methods. The combination scenario was selected according to the following criteria:

1. the total required data should still fit on the memory of a single socket for all machines considered, i. e., a maximum of 48 GB

2. the number of component grids should be large enough to allow to distribute them to a significant number of process groups,

3. the minimum level should be sufficiently high to allow for meaningful domain decompositions even on the smallest grids.

These three goals can be in conflict, as criterion 2 calls for a big difference $\vec{L}$ between $\vec{\ell}^{\min}$ and $\vec{\ell}^{\max}$, while criterion 3 calls for both to contain high values and criterion 1 needs both to contain low values (which can be alleviated by a larger difference).

Fortunately, a suitable combination scheme could be identified in this search space:

Combination scheme for strong scaling

$\vec{\ell}^{\min}$	$(2,3,3,3,3,3)$	# grids	923
$\vec{\ell}^{\max}$	$(8,9,9,9,9,9)$	# finest grids	462
total FG #DOF	5.27×10^{9}	mem. finest grids	64 MiB
total FG memory	39 GiB	#DOF FG at $\vec{\ell}^{\max}$	9.01×10^{15}

To distribute the scheme among process groups—in the absence of a manager rank—a static load balancing based on a balanced number of DOF per process group was used. This means, that for higher numbers of process groups, it becomes very unlikely for a process group to hold a cluster of main diagonal grids, and the ChunkedOutgroupSGReduce effectively becomes a chunked SGReduce, cf. Sections 4.4.3 and 4.4.4. The chunk size of 128 MiB per thread was chosen such that the communication buffers still fit into the socket's main memory for all machines and parallelizations considered. Of course, selecting this relatively small scheme that fits into 48 GB of memory means that the number of DOF per process becomes very small for the higher degrees of parallelism. For instance, on HAWK's up to 2^{19} cores, each thread has to compute only $\approx$ 10,048 DOF. In this limit, it becomes apparent that overheads and synchronization will dominate the run time, and one can expect the parallel efficiency to drop significantly.

For the measurements, each scenario was run once and 11 solver 'runs' of 10 solver time steps each were performed, and 10 intermediate combinations in between these solver updates. As shown in [137], more time steps per combination are possible for the advection solver at very little loss of accuracy. There were `MPI_Barriers` introduced after the run and combine steps, such that potential load imbalances in one of the steps would not affect the measurements of the other. Also, since no I/O is included in the measurements, the best measure of success is the maximum run time of any process. The experiments were performed in July-September 2023 with commit `ba519e39` of DisCoTec, except for the simulations with $n_g > 256$, which used commit `e5cffee2`.

The approach to data point selection was to scale up in both ways, starting from one rank: One single process groups that gets larger and larger (vertically in Figure 4.1), as well as process groups of one that get more and more numerous (horizontally). For the higher process counts, $n_g = 64$ groups and $n_p = 256$ were selected for further scaling up, respectively—the motivation being that the combination's parallel efficiency had dropped to a similar extent on Fritz, JUWELS, and SuperMUC-NG for $(n_g, n_p) = (64, 1)$ and $(1, 256)$. This approach results in a 'rectangle' of measurements in the space of parallelizations $(n_g,\ n_p)$, where the lines meet again at $(64, 256)$ and may be extended 'upwards'.

Figures 5.5 to 5.8 show the maximum time of any rank for the solver and combination steps, respectively, and Figures 5.9 to 5.12 show the respective strong scaling efficiencies. To give a sense of the 'height' of each data point, the point markers are encoded by color values, while the line and area colors as well as the marker shapes represent the respective system. The scales and color bars are set uniformly on all of Figures 5.5 to 5.8 as well as Figures 5.9 to 5.12, respectively (an exception to this is the z axis of HAWK's interesting combination efficiency). One can observe that—across all systems—the time spent on the solver step drastically decreases for higher parallelizations, just as expected. For the combine step, the behavior is similar for larger process groups, but for more process groups, the time spent on the combine step stagnates and may even increase again for higher group counts n_g. This is where the allreduce's complexity of $\mathcal{O}(\log(n_g))$ starts to dominate the combination time.

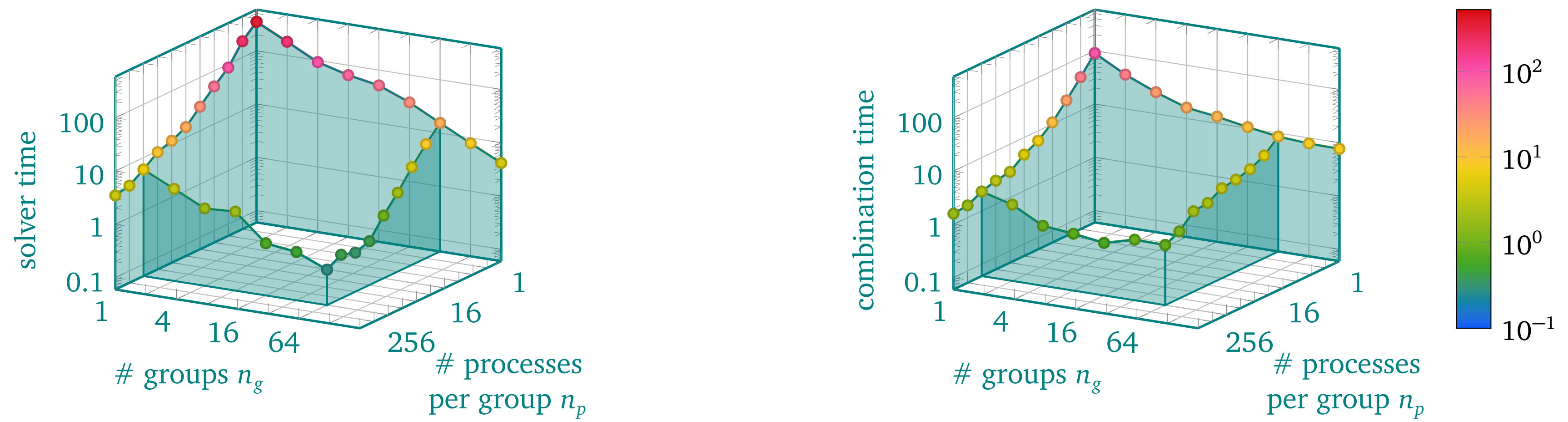


Figure 5.5.: Strong scaling timings on Fritz: Maximum time required on any rank.

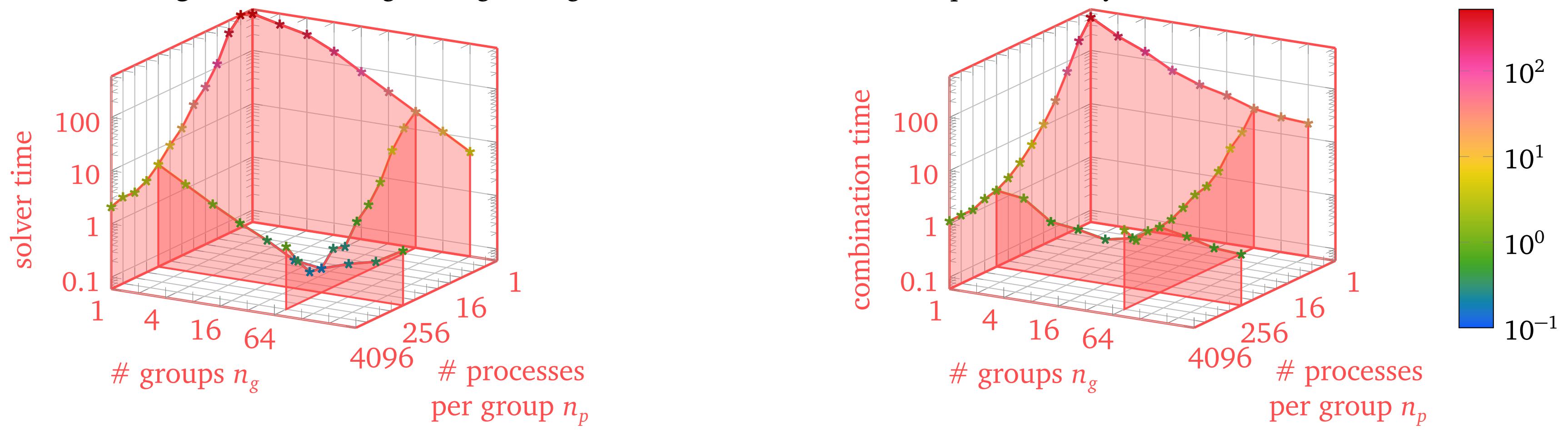


Figure 5.6.: Strong scaling timings on HAWK: Maximum time required on any rank.

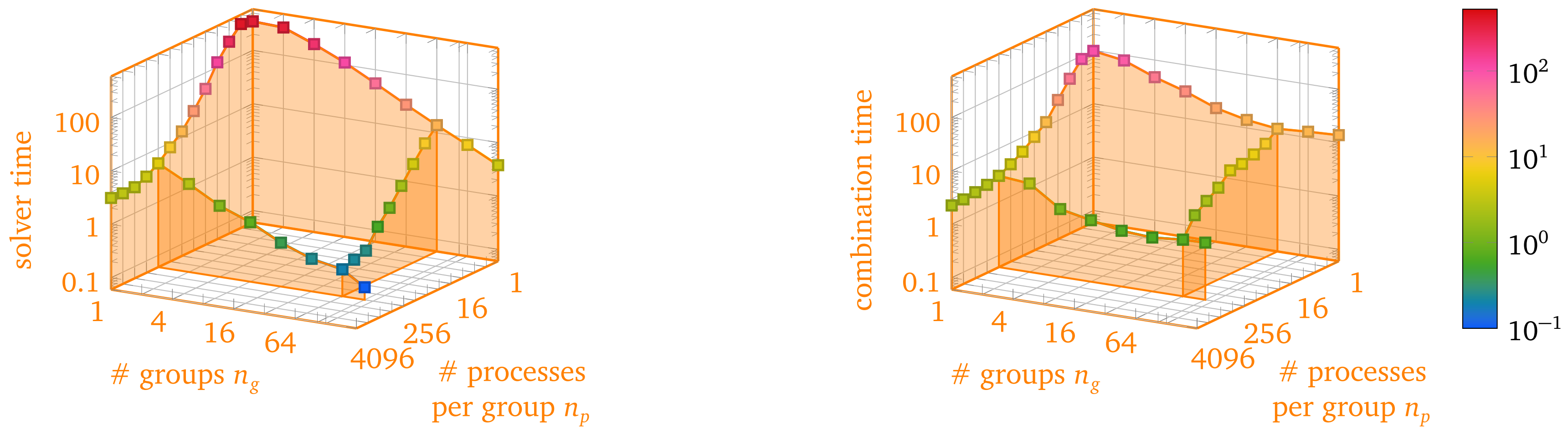


Figure 5.7.: Strong scaling timings on JUWELS: Maximum time required on any rank.

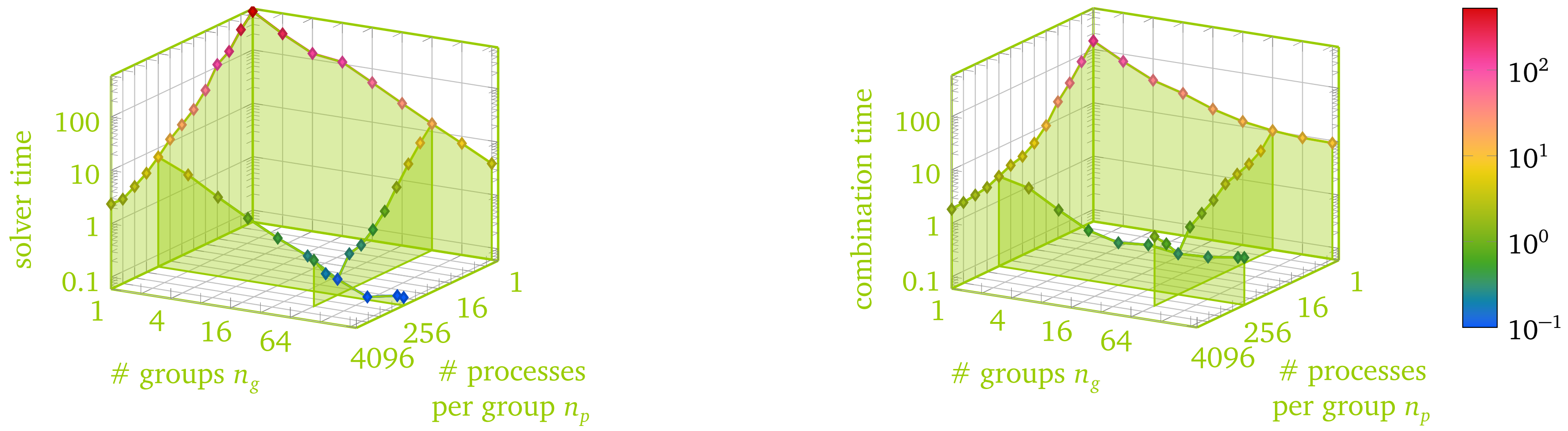


Figure 5.8.: Strong scaling timings on SuperMUC-NG: Maximum time required on any rank.

For the JUWELS system, one can observe that the timings decrease less than for the other systems, which is particularly unexpected for the lower degrees of parallelization (that still operate on a single node). The effect is more clearly visible from the strong scaling efficiencies in Figure 5.11, which immediately drop below 70%. This effect could be attributed to wrong process placement and pinning through SLURM on JUWELS [140], which led to imbalanced assignments of ranks to sockets and threads to cores. The error could only be resolved, thanks to help from Thomas Breuer at the JSC, for the two highest-parallelized data points: 64 and 106 process groups of $n_p = 256$ ranks. Unfortunately, there was not enough leftover compute time in the JUWELS allocation to recreate the other measurements with the corrected pinning. Note that the 106 groups occupy (almost) the entire JUWELS system.

Another system with peculiar scaling behavior is the HAWK system: From Figure 5.10, one can see an enormous increase in combination efficiency for larger process groups, with efficiencies higher than 2. The reason for these unexpectedly high efficiencies is rooted in the highly strided memory accesses for the basis transformations, as well as the architecture of HAWK's AMD EPYC Zen2 CPUs: As mentioned in the context of OpenMP threads, only four cores (here, one DisCoTec rank) shares a L3 cache of 16 MiB. However, the largest grids occurring in the combination scheme are of size 64 MiB, the second-largest are of size 32 MiB, and so on. Introducing a finer domain decomposition, which is equivalent to larger process groups, leads to implicit cache blocking [70] in the combination step: Since the six-dimensional full grid tensor is not the only data required in memory, one can assume that for a single process, only the 8 MiB grids may fit entirely into the L3 cache for the repeated basis transforms into the different dimensions. Only at sixteen ranks, the largest grids in the combination scheme start to comfortably sit in the L3 cache, where the maximum efficiency ϵ_s of > 2.3 is reached, and the efficiency starts to decrease again. The tremendous influence of caching on the basis transformations was discussed by several previous works [74, 84], and DisCoTec currently only addresses it with distributed-memory parallelism. Looking at the scaling behavior on HAWK, it could be worth considering how the shared-memory cache optimizations [84] can be integrated with the current approach. Another way to re-use the caches in the basis transformation step could be to use temporal blocking by reversing the iteration direction between the different dimensions, cf. [169].

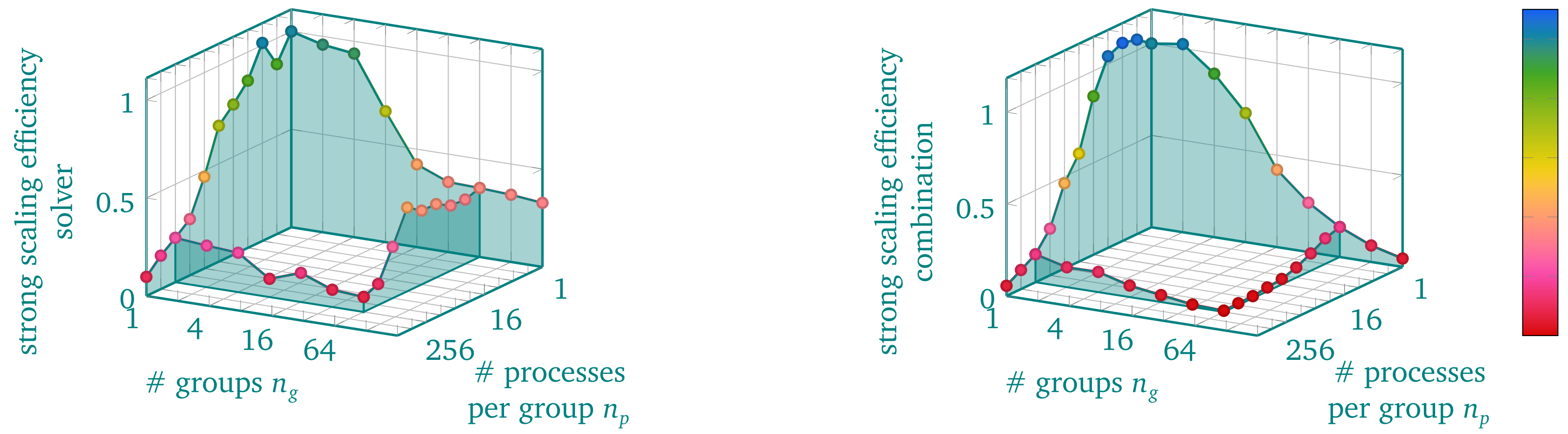


Figure 5.9.: Strong scaling efficiencies on Fritz

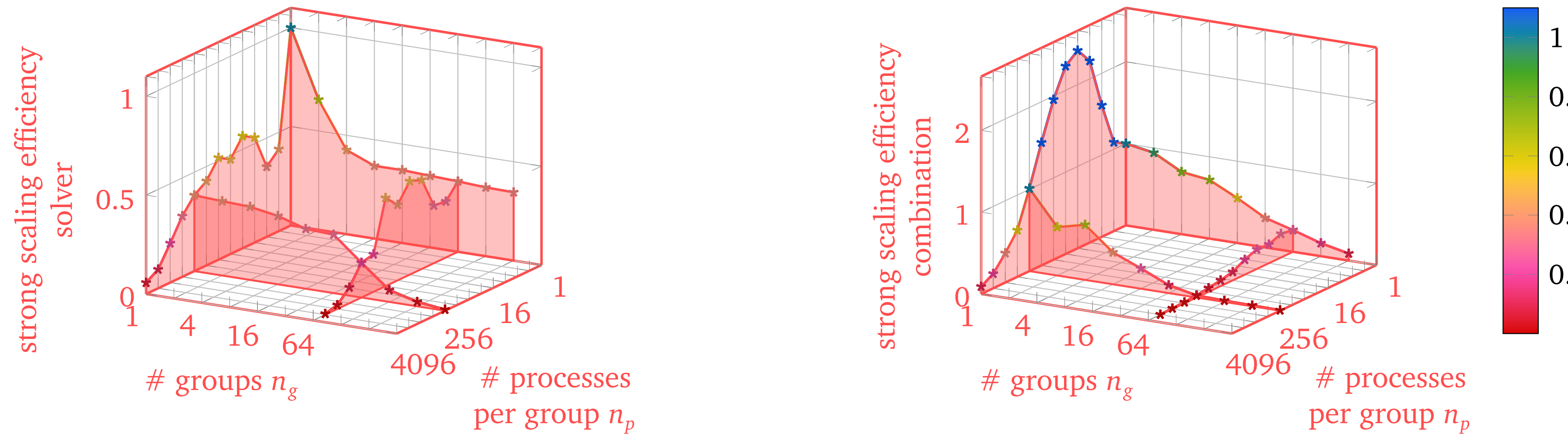


Figure 5.10.: Strong scaling efficiencies on HAWK

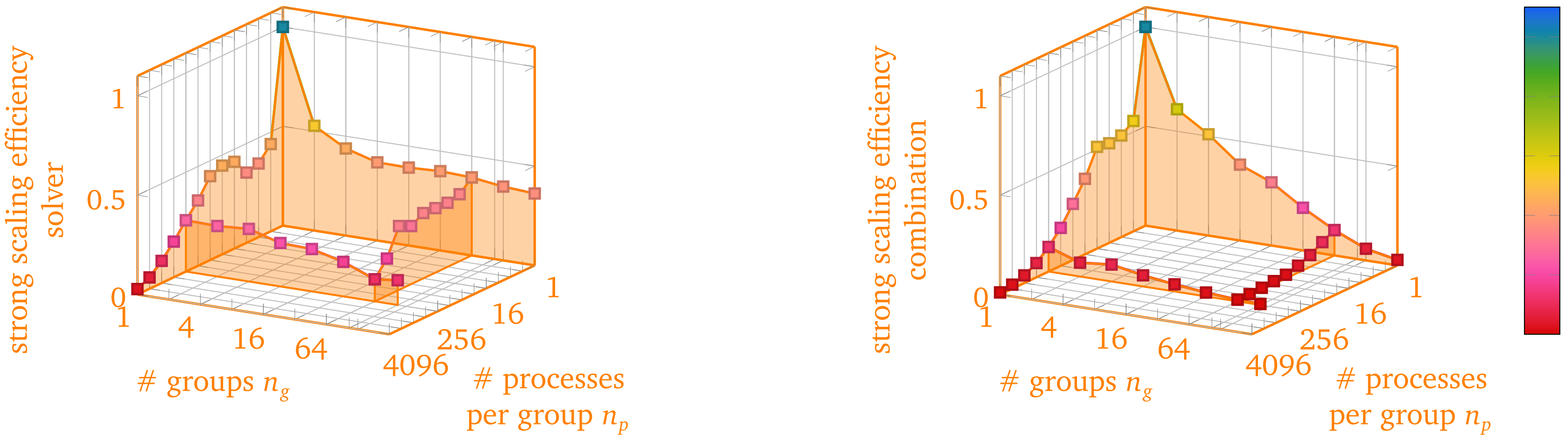


Figure 5.11.: Strong scaling efficiencies on JUWELS

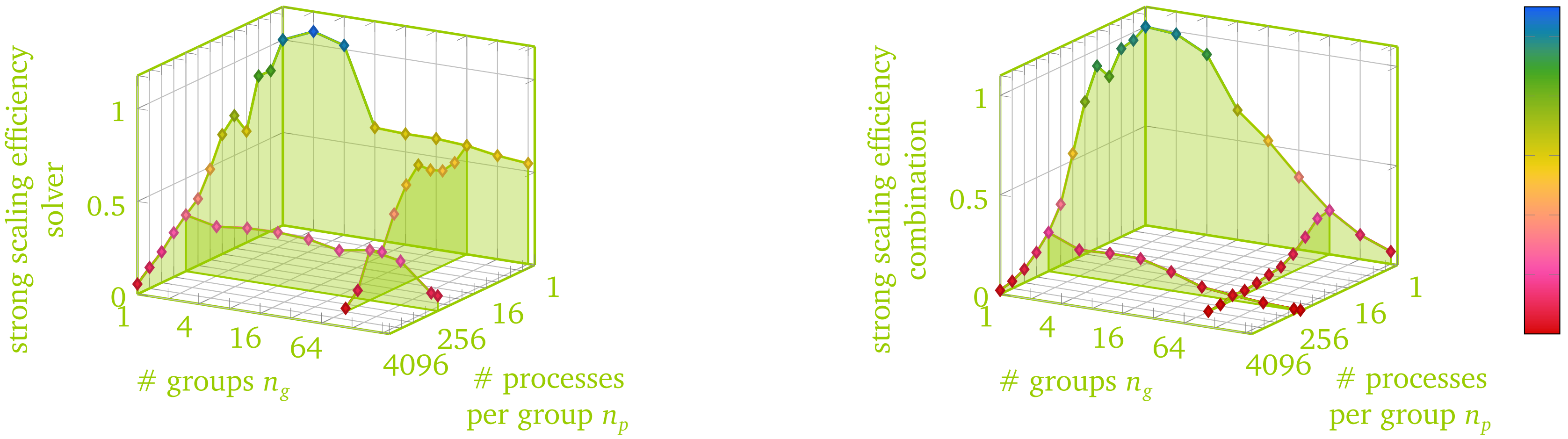


Figure 5.12.: Strong scaling efficiencies on SuperMUC-NG

Lastly, of course, there is a trivial way to achieve finer cache blocking in DisCoTec: Instead of using OpenMP threading, one could use a higher number of MPI ranks in the simulation again, to achieve the implicit cache blocking effect.

A similar, but much smaller hump is visible for the efficiency of the combination step for the Fritz system, cf. Figure 5.9, which has L3 caches of 54 MiB per socket, the same as SuperMUC-NG. Likely, the effect would also be visible for SuperMUC-NG in Figure 5.12 if the total time required for the combination was not as high for low degrees of parallelism.

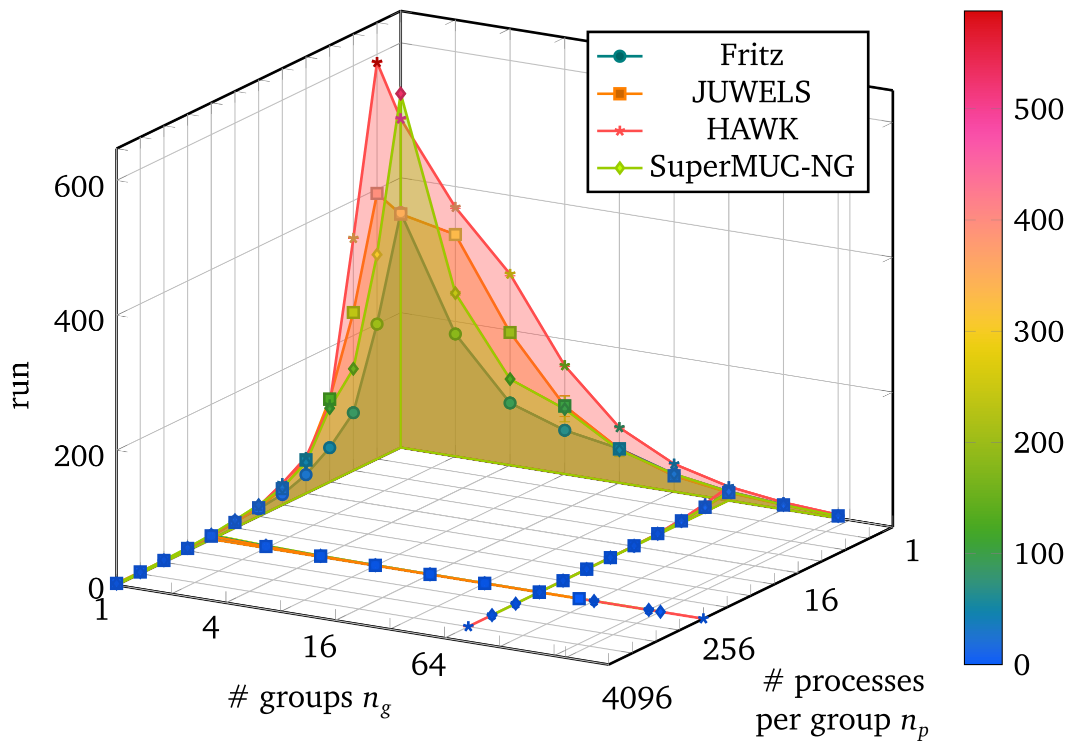
This is better illustrated by the direct comparison: Figure 5.13 shows the linear-scale timings for the solver update and combination steps on all systems—the same data as in Figures 5.5 to 5.8—in a single plot.

One can see that, for the lower degrees of parallelism, the time spent on the solver update step is significantly higher than for the combination step (apart from HAWK, as already discussed). From the plotted timings and the corresponding parallel efficiencies however, we can extrapolate that this will not be sustained through the higher degrees of parallelism, as the combination is not entirely as embarrassingly parallel between the process groups as the solver step.

While the relatively high time measurements for HAWK at low degrees of parallelism can be explained by the aforementioned cache effects, and for JUWELS by the incorrect pinning—which apparently affected the solver step more than the combination step—there is no substantial explanation why the run time on SuperMUC-NG is visibly high for low degrees of parallelization: In the case of a single rank ($n_g = 1, n_p = 1$), both the solver and combination steps on SuperMUC-NG take approximately 1.5 times as long as on Fritz and JUWELS, and not only in the maximum, but also in the mean and median statistics.

Comparing the timings of the different parts of the combination step in Algorithm 4.4, the overhead of ≈ 1.5 for SuperMUC-NG is evenly distributed between the basis transformations and the reduction.[1] Looking at Table 5.1, the only notable differences between SuperMUC-NG and the other systems are the clock rate and the type of interconnect. The interconnect, however, is going to play a role for simulations running on more than one node, which is only the case for > 12 ranks on SuperMUC-NG. The marginally lower CPU clock frequency (2.3 GHz without EAR for SuperMUC-NG compared to 2.4 GHz on JUWELS and 2.7 GHz on Fritz) is

[1]Quite different for HAWK, where the reduction virtually behaves the same way as for the other systems, and only the basis transforms introduce overheads of factors > 7 compared to Fritz and JUWELS.



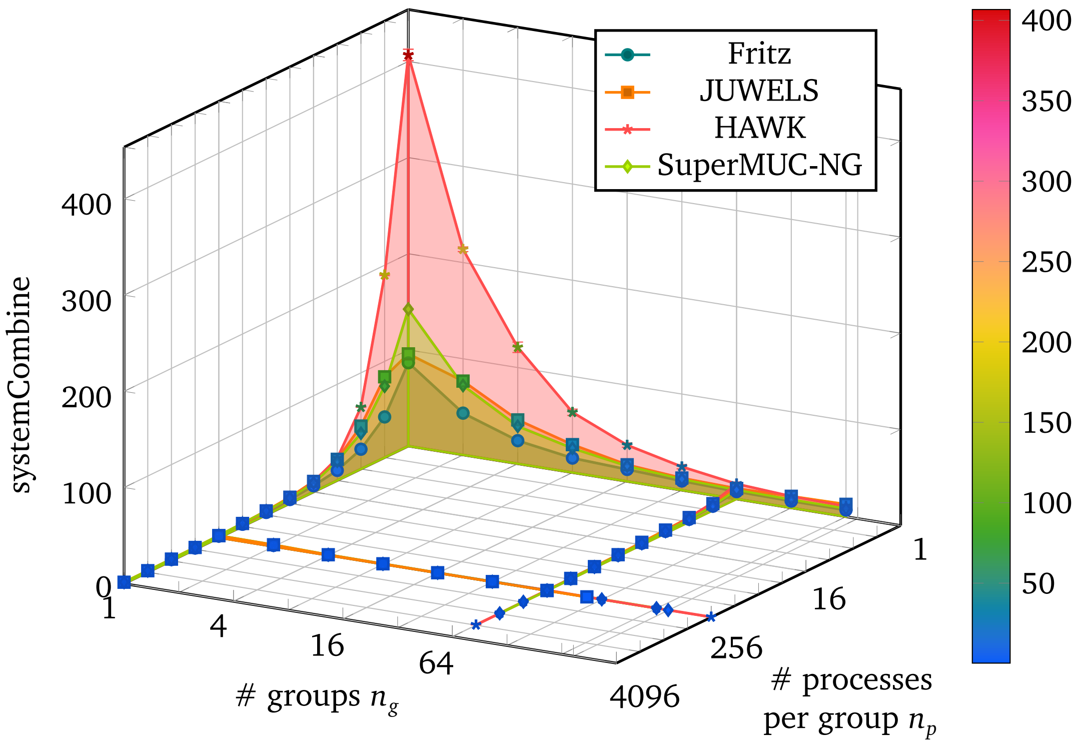


Figure 5.13.: Strong scaling timings on all four machines, linear scale, for comparison: Maximum time required on any rank. Colors and markers are set the same way as in Figures 5.5 to 5.12.

also not enough to account for this difference; In fact, since the solver step should be bandwidth-limited for the lower degrees of parallelization, the clock rate is expected to be even less relevant.

However, if we also consider the higher parallelizations—which are not well distinguishable on a linear scale [69]—we can observe that SuperMUC-NG performs its computations quicker than the other machines. At the parallelization ($n_g = 64, n_p = 256$), which is where the two lines of higher parallelizations meet, the situation is quite different, and the maximum time measured for the solver update is 0.19 s on JUWELS, 0.17 s on HAWK, and only 0.12 s on SuperMUC-NG. This is also reflected in the substantially higher parallel efficiencies for SuperMUC-NG at higher process counts. One can hypothesize that SuperMUC-NG's hardware and/or software stack are better geared towards higher degrees of parallelism, at the cost of slight losses for low-parallelized jobs. But the exact reason for the higher run times on SuperMUC-NG for all measured parts of the simulation for *lower* degrees of parallelism, as well as the lower run times for *higher* degrees of parallelism still remains an open question.

Overall, it is possible to execute DisCoTec simulations on up to full CPU-based HPC system sizes, while the run time-to-solution generally decreases for higher process counts. Still, the strong scaling efficiency deteriorates for higher parallelizations, just as expected for this relatively small combination scheme.

5.4.2. Weak Scaling Along the Memory Limits

The aim of this section is to analyze DisCoTec's scaling behavior on the memory limit through weak scaling, to complement the strong scaling measurements. Accordingly, the design criteria for the weak scaling setup were different from the strong scaling:

- The memory allotted to component grid data structures (for holding the distribution function f) should be approximately 1 GiB per core when the entire scheme is distributed to 64 process groups.
- The scheme should have a high and uniform level difference $\vec{L} = \vec{\ell}^{\max} - \vec{\ell}^{\min}$.
- The minimum level should be sufficiently high to allow for meaningful domain decompositions, even on the smallest grids. In particular, the advection solver needs at least two points ($\hat{=}\ \ell_j \geq 1$) in each dimension j to compute its derivatives.

This resulted in the following sequence of scenarios, which can be run on parallelizations (n_g, n_p) of $(1, 4)$ to $(64, 2048)$.

Combination scheme for weak scaling, groups of $n_p = 4$

$\vec{\ell}^{\min}$	$(1,1,1,1,1,1)$	# grids	66,605
$\vec{\ell}^{\max}$	$(17,17,17,17,17,17)$	# finest grids	20,349
total FG #DOF	1.37×10^{11}	mem. finest grids	32 MiB
total FG memory	1018.82 GiB	#DOF FG at $\vec{\ell}^{\max}$	5.07×10^{30}

...

Combination scheme for weak scaling, groups of $n_p = 2048$

$\vec{\ell}^{\min}$	$(3,3,3,2,2,2)$	# grids	66,605
$\vec{\ell}^{\max}$	$(19,19,19,18,18,18)$	# finest grids	20,349
total FG #DOF	7.00×10^{13}	mem. finest grids	16 GiB
total FG memory	509.41 TiB	#DOF FG at $\vec{\ell}^{\max}$	2.60×10^{33}

Furthermore, the file-based combination should be tested and its impact on the run time measured; At the same time, the assignment to process groups should be optimized for low data transfer volumes with ChunkedOutgroupSGReduce (cf. Algorithm 4.5). To achieve these two effects, the scheme $\mathcal{I}^{\text{CT}}$ was first split with the level-sum criterion Equation (4.10) into two (almost) equal partitions. Then, each of these partitions was assigned to 32 groups with the METIS-based heuristic partitioning, cf. Section 4.6.2, and the 64 resulting subsets were assigned to the (up to) 64 groups. This means that, as part of each combination, the conjoint data between the two initial partitions is written out and read back in, for measurement of the I/O timings. (As part of a widely-distributed simulation with these partitions, a reduction operation would take place at this stage of the algorithm.) The resulting sizes of the conjoint sparse grid for this case can already be considerable: For the $n_p = 4$ scenario, the conjoint sparse grid contains 500,037,632 DOF or 3.73 GiB, and by the increase of resolution up to $n_p = 2048$, one obtains 256,019,267,584 DOF or 1.86 TiB of conjoint data. (Note that the conjoint size required on ranks of the I/O group will always be 125,009,408 DOF or 953.75 MiB per rank, or 238.44 MiB per core). Since the scenario design limits us to 64 groups, the higher process group size n_p was selected differently on each of the systems, such that 64 groups of this size would just fit if run on the system. This means that the higher n_p will be 256 for Fritz and JUWELS, 1024 for SuperMUC-NG, and 2048 for HAWK.

The ChunkedOutgroupSGReduce algorithm used chunks of 64 MiB per core, or 256 MiB per rank ($n_t = 4$). Like in the strong scaling scenario, there were ten solver time steps per combination step. This time, five intermediate combinations were performed to save on core-hours, which are otherwise quickly used up at the higher degrees of parallelization with weak scaling. Some further exceptions apply to the measurements with the highest degrees of parallelism on JUWELS and HAWK: The simulation for ($n_g = 64, n_p = 256$) on JUWELS performed three combinations with one solver time step (figures extrapolated for the plots), and the simulation with ($n_g = 64, n_p = 2048$) on HAWK could only complete two solver updates—and one full combination in between—within the allotted time frame.

The graphs in Figures 5.14 to 5.17 show the timings measured for the solver update and combination parts on each system, respectively. The marker color scale is the same for all plots, and the linear range is constant within each of the parts (solver and combination). The plots show the mean times with standard deviations across the steps and all ranks. High variances in the solver timings can therefore already indicate some load imbalance. To obtain the time for the combination without I/O, only the basis transformation and the Reduce/Scatter parts of Algorithm 4.1 (i. e., the combination without file system interactions and the following broadcast as described in Section 4.5.3) were considered. The weak scaling efficiencies displayed in Figures 5.18 to 5.21 are computed as in Equation (5.3), dividing the mean time required on $n_g \cdot n_p$ ranks by the mean time required on ($n_g = 1, n_p = 4$) ranks.

It can be seen from Figures 5.16 and 5.20 that some data points are missing for JUWELS. These jobs were not able to execute as they ran out of memory on some nodes, which can be attributed to the wrong pinning discussed in Section 5.4.1. Again, the JUWELS measurements should generally be taken with a grain of salt because of the pinning error, which was only corrected for the highest measured degree of parallelism at ($n_g = 64, n_p = 256$).

Overall, one can observe that the parallel efficiencies are now significantly higher than in the strong scaling case, since the ‘work’ to be done per rank now stays approximately constant.

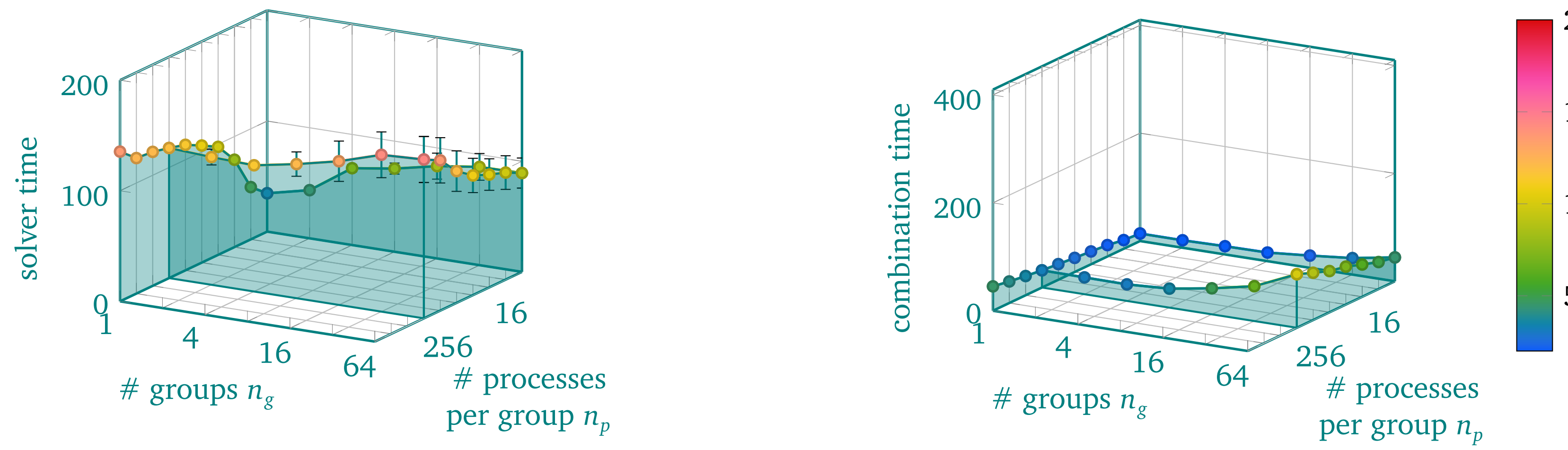


Figure 5.14.: Weak scaling timings on Fritz: Mean times with standard deviations across ranks and steps.

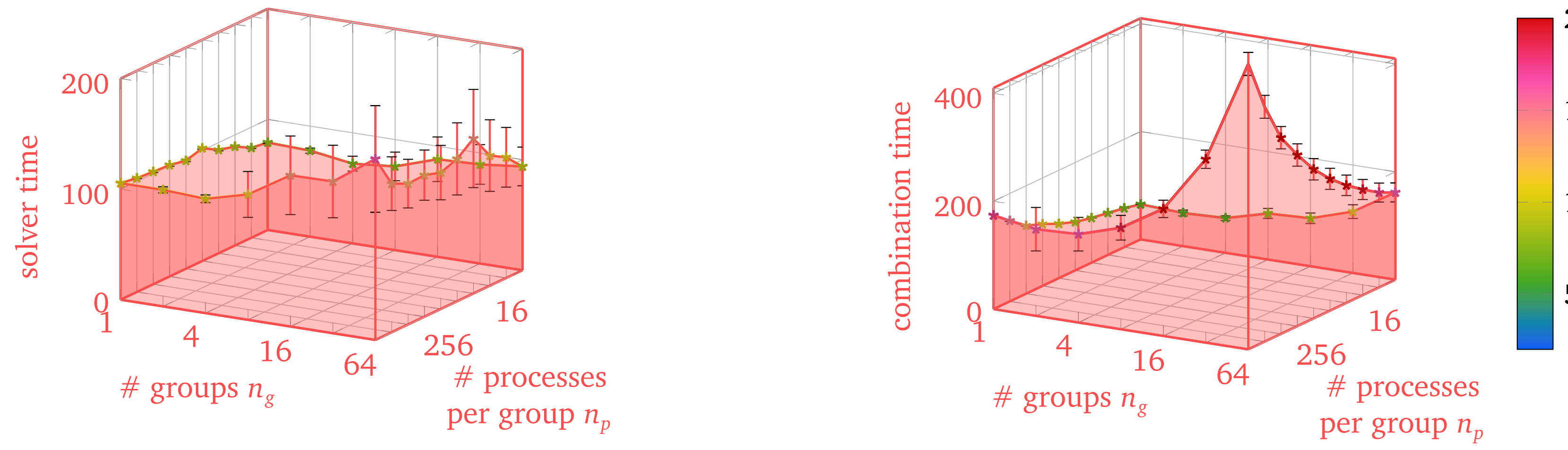


Figure 5.15.: Weak scaling timings on HAWK: Mean times with standard deviations across ranks and steps.

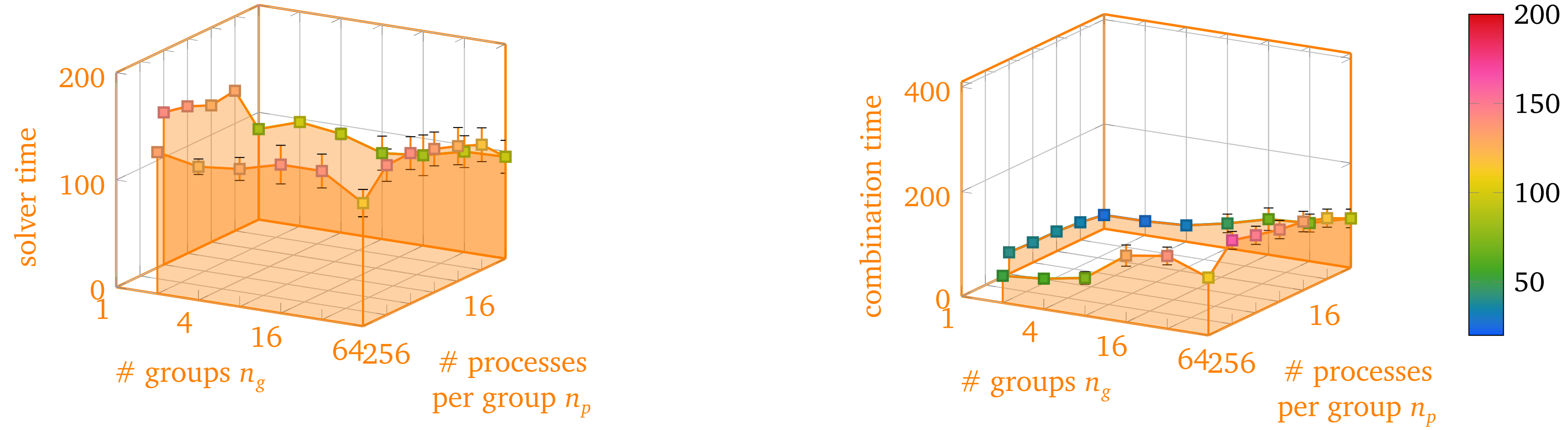


Figure 5.16.: Weak scaling timings on JUWELS: Mean times with standard deviations across ranks and steps.

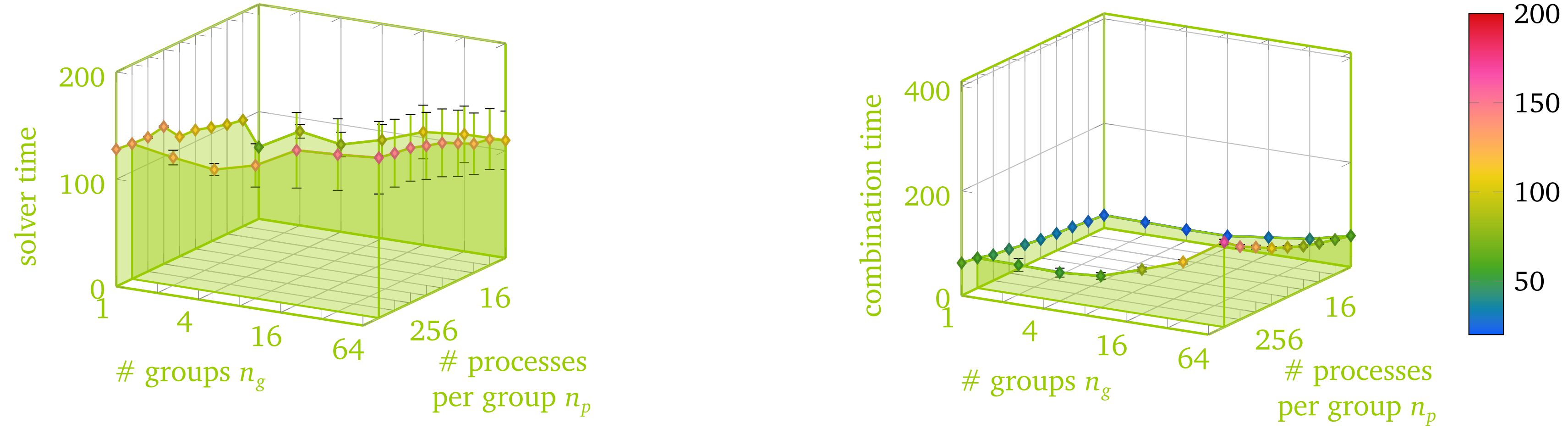


Figure 5.17.: Weak scaling timings on SuperMUC-NG: Mean times with standard deviations across ranks and steps.

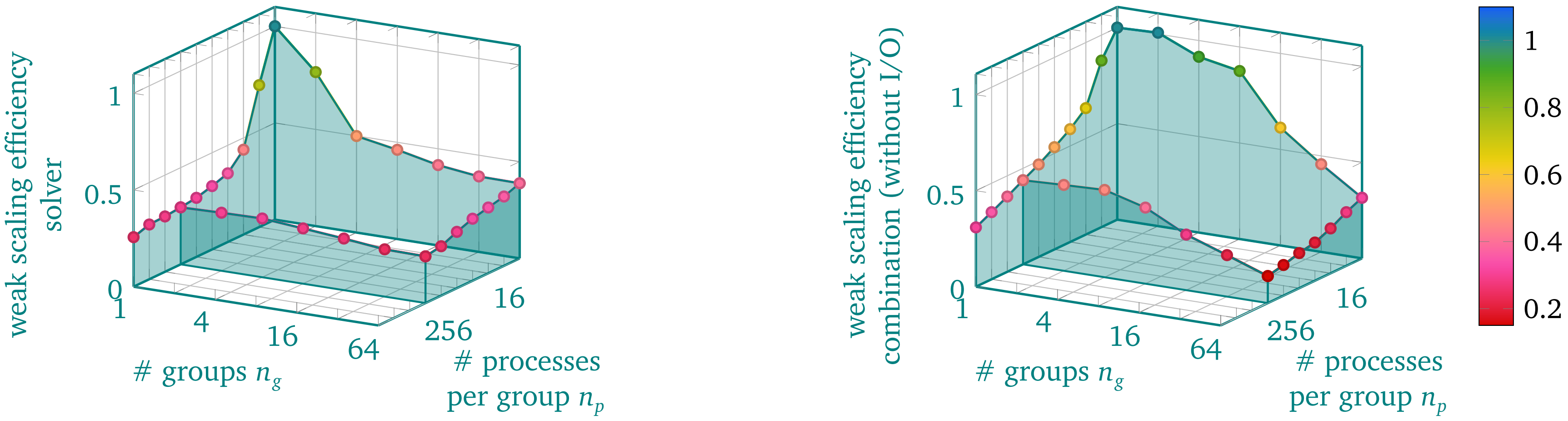


Figure 5.18.: Weak scaling efficiencies on Fritz

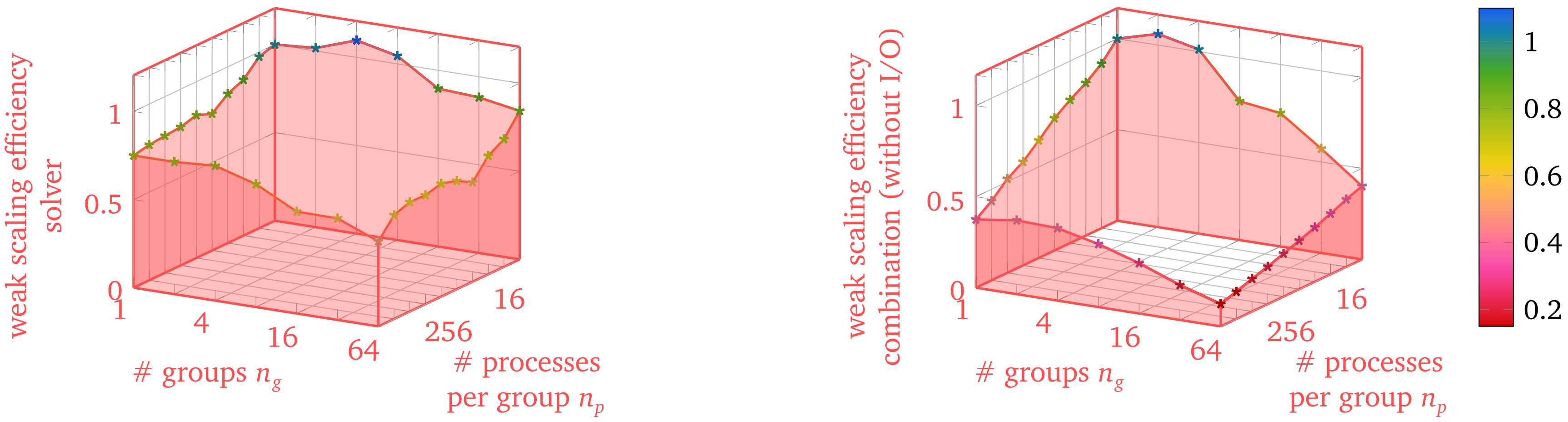


Figure 5.19.: Weak scaling efficiencies on HAWK

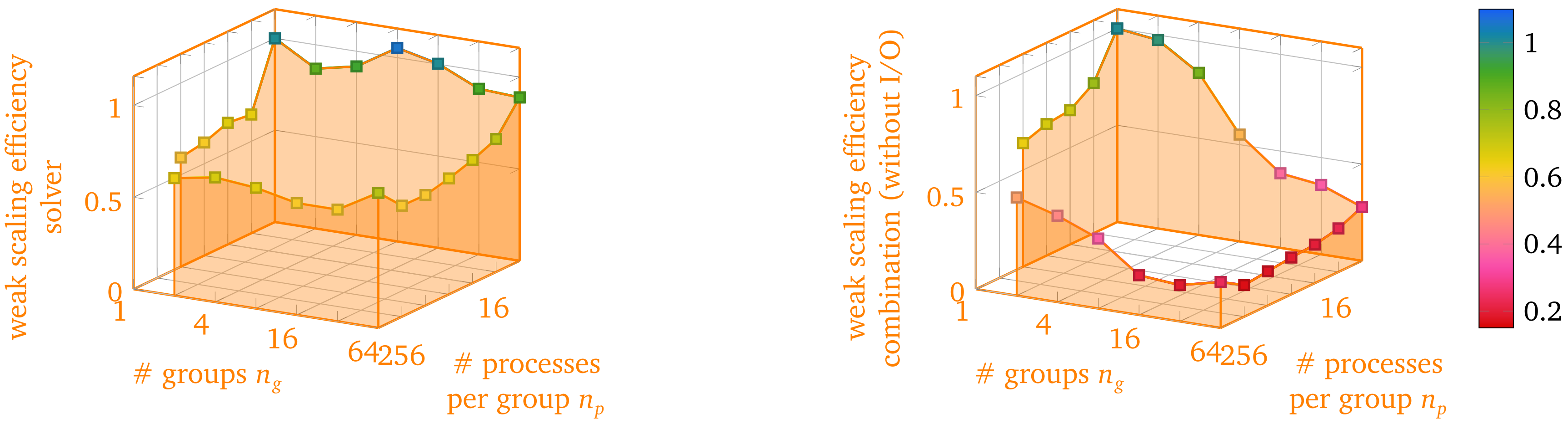


Figure 5.20.: Weak scaling efficiencies on JUWELS

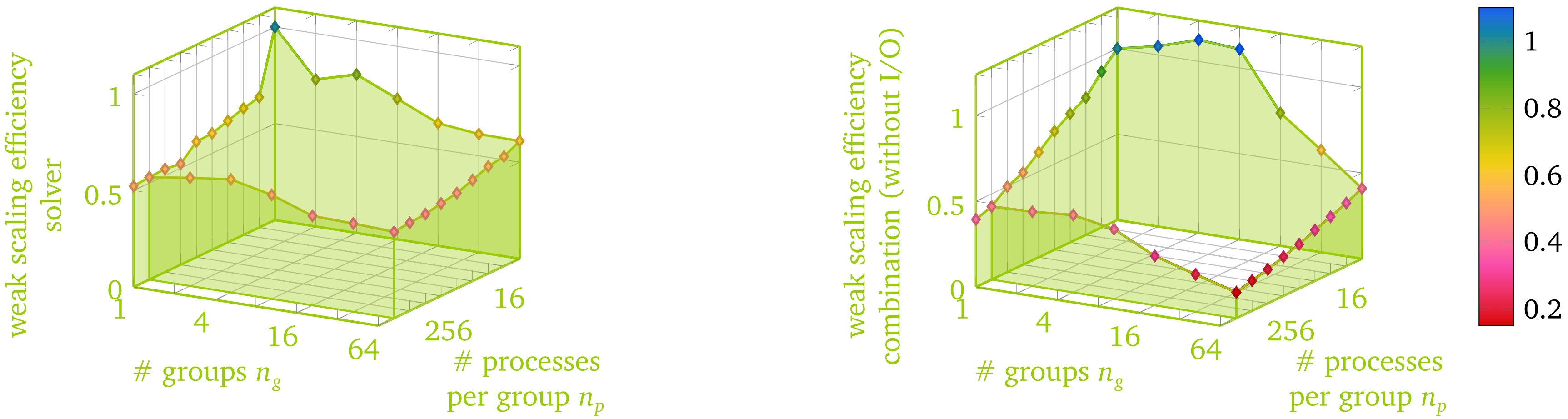


Figure 5.21.: Weak scaling efficiencies on SuperMUC-NG

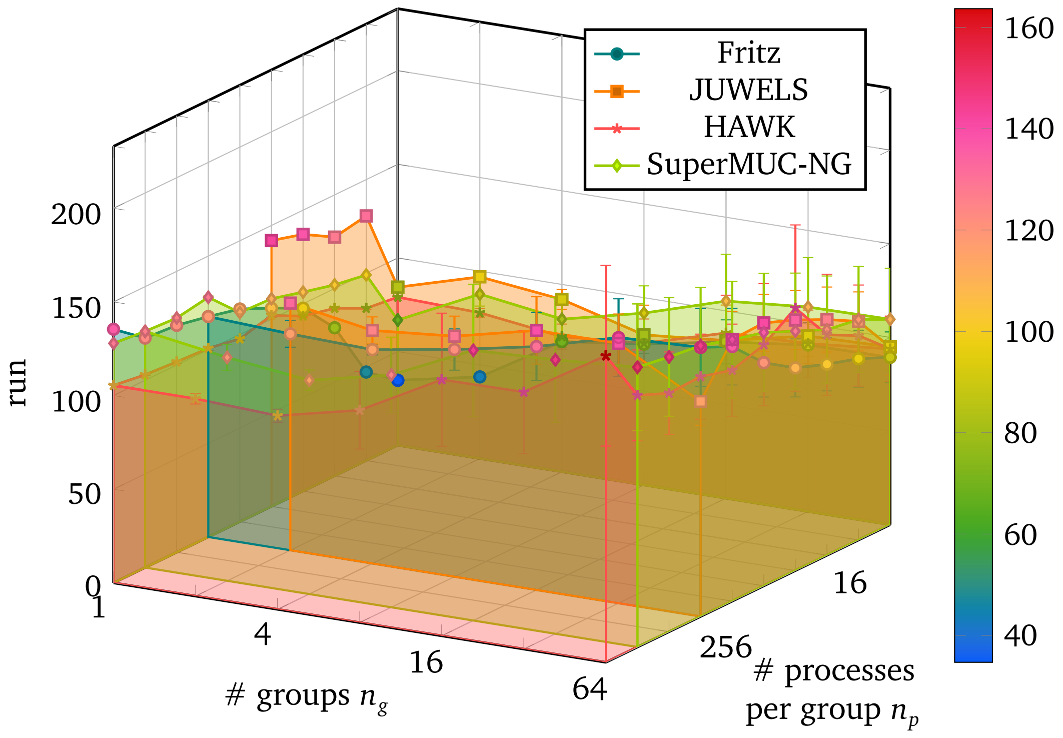
In particular, the solver update step tends to have a higher parallel efficiency than the combination step. For the *solver update step*, the JUWELS system retains parallel efficiencies as high as 60%, HAWK can retain 48% or more, and SuperMUC-NG still has 45% of parallel efficiency.

Fritz's relatively low 24% of parallel efficiency in the solver step can be explained by the fact that the solver is comparably fast on a single rank already (possibly due to the high clock rate, high memory bandwidth, and/or relatively large L3 cache per socket). Consequently, the MPI communication overhead hits hard by comparison. Still, the solver update step on Fritz at a parallelization of $(n_g = 64, n_p = 64)$ takes only 119.14 s, which is less than on the other systems considered. Looking at the 'highest parallelized' data point common to all systems, $(n_g = 64, n_p = 256)$, the solver update step is performed fastest on the JUWELS system (with corrected pinning) at 92.30 s.

The minimum parallel efficiency for the *combination step* on the considered systems is obtained for the highest-parallelized problems and approaches values of $\approx 12-19\%$. This observation comes of course with the qualifier that the combination timings on HAWK are relatively long, due to the already observed effect of the L3 cache misses. The cache miss effect actually hits HAWK twice as hard in the weak scaling scenario: Again, the strided accesses in the basis transformation step require that cache lines have to be loaded again for the different dimensions of basis transforms according to the unidirectional principle [162]. But added to this, the horizontal reduction between the process groups uses chunks of 256 MiB per core complex, where the available L3 cache for the addition operation has only 16 MiB. (The second effect was not yet visible in the strong scaling scenario as the communication volume between groups would decrease for higher process counts.) Excluding HAWK, the minimum parallel efficiencies for the combination step range from $\approx 15 - 19\%$.

Note that the $\mathcal{O}(\log n_g)$ scaling of the allreduce operation, which was a limiting factor for the parallel efficiency in the strong scaling scenario, is generally not significant in the weak scaling scenario, since the actual time spent on the allreduce is low compared to the other parts of the combination step—as long as each core has sufficient work to do in the hierarchization and dehierarchization as well as data gathering and scattering between the SG and the component grids.

For a direct comparison between systems, Figure 5.22 shows the timings for the solver update and combination steps—the same data as in Figures 5.14 to 5.17—on all systems in a single plot. Most notably, the mean solver timings are relatively



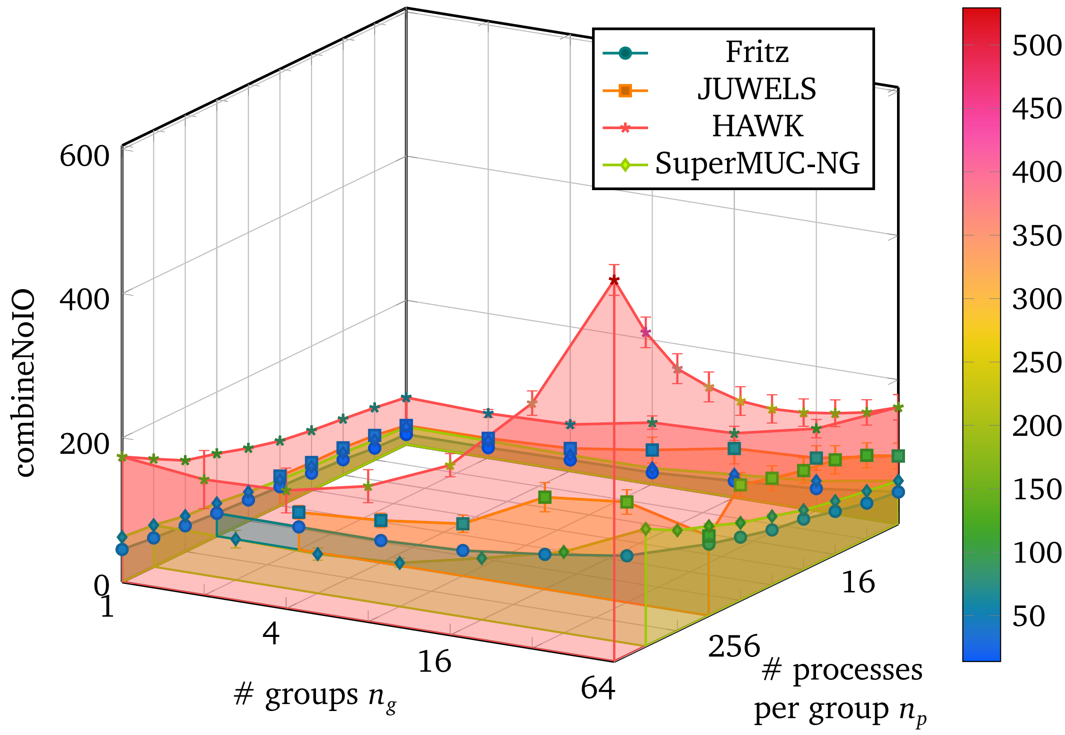


Figure 5.22.: Weak scaling timings on all four machines, linear scale, for comparison: Mean times with standard deviations across ranks and steps. Area colors and marker shapes are set the same way as in Figures 5.14 to 5.21.

similar between the systems and stay rather constant between the different degrees of parallelism.

The lower parallel efficiencies of the combination step should be tolerable, especially if we consider that in some scenarios, it is possible to run many run steps per combination step without a loss in accuracy, and also that typical Vlasov solvers will have a higher computational complexity and run times than the advection solver used here, cf. GENE in Section 5.3.1. For this setting already, with a relatively simple solver and ten solver steps per combination, the timings of the combination part are lower than the solver updates (with HAWK forming an exception here for the higher core counts, and SuperMUC-NG only for the three highest parallelized runs).

5.4.3. Load Imbalance Through Communication-Optimized Grid Assignment

On the 'right' of the solver timing plots (the respective first plot in Figures 5.14 to 5.17 and 5.22), which corresponds to 64 groups, one can observe relatively high standard deviations (as indicated by the error bars) in both the solver and combination steps. This is due to a load imbalance between the process groups, which can be seen from Figure 5.23, where the load imbalances for each measured scenario are computed according to Equation (5.4).

By contrast, on a single process group, the load appears perfectly balanced. This is expected: Even if one of the cores would, by random chance, take very long to compute its part of the solver update, the imbalance would be hidden by the necessary implicit synchronization within each component grid's solver step, which 'drags' all the other ranks in the group to take the same time.

One can observe that the load imbalance worsens significantly when going from four to eight process groups, and further increases for 16 process groups, with load imbalances of 1.6 and higher. This is an indication that the group assigned the highest load is somewhere between group numbers 9 and 16. For the right side of the plot, the load imbalance tends to increase, which is due to the tendency of the mean solver update time to increase, as can be seen in Figure 5.22.

However, these effects were not observed for the strong scaling scenario (cf. Figures 5.5 to 5.8 and 5.13), where the load imbalance was significantly lower—although, in theory, the scarcity of work could actually worsen the load imbalance. A significant difference between the two scenarios was the 'naive' DOF-only assignment of tasks to process groups for the strong scaling versus the METIS-optimized assignment for the weak scaling. As discussed in Algorithm 4.4, the optimized

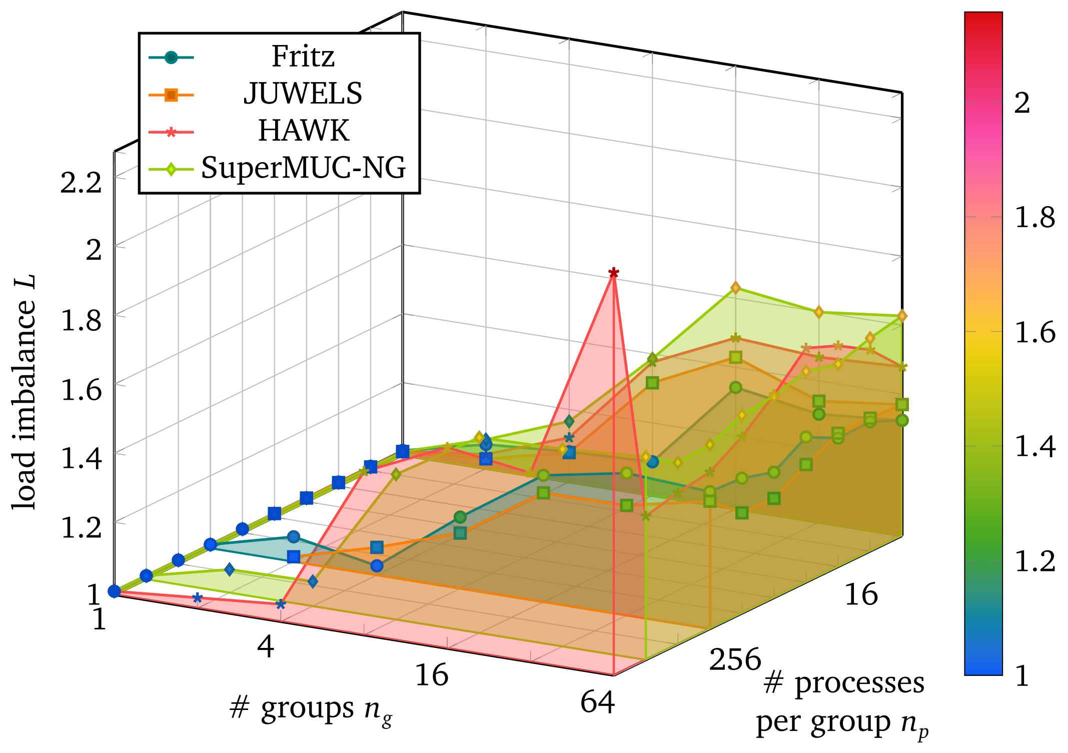
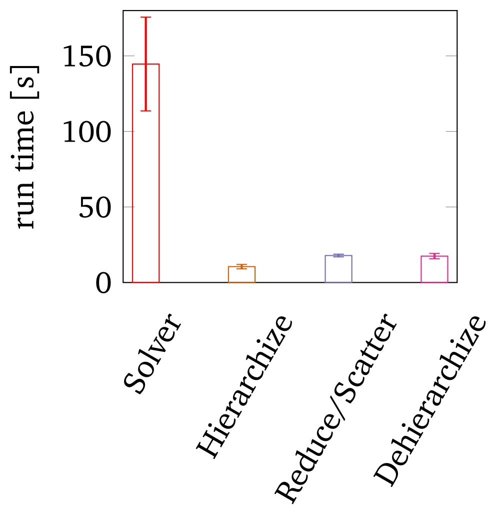


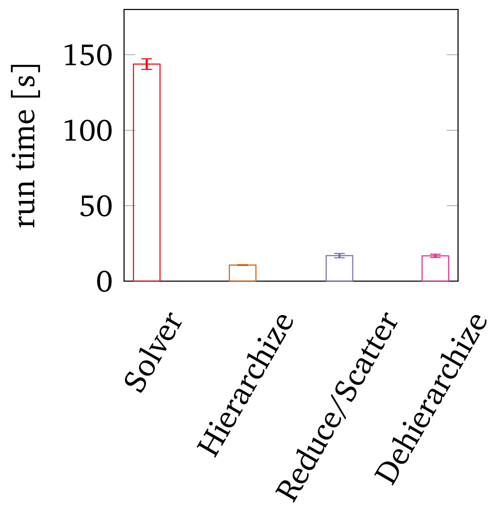
Figure 5.23.: Load imbalance, cf. Equation (5.4), for the same data points as present in Figure 5.22.

assignment can lead to significant reductions in the communication volume of the allreduce between process groups in the combination. But evidently, the same optimization can lead to imbalances for the solver update step.

To validate this proposition, one of the simulations was re-run with the naive assignment. Figure 5.24 shows a comparison of the run times for the two different assignment strategies. In fact, the variance is visibly reduced, while the mean time stays approximately the same for the solver update scheme. This difference can be explained by the contiguous patches of the component grids in the combination scheme that is generated by the METIS-based assignment: The group that is assigned a corner of the simplex will also be assigned the adjacent grids and further neighbors—and the corners as well as their neighbors have the highest anisotropies of all the grids in the combination scheme $\mathcal{I}^{\text{CT}}$. High anisotropies in the grid lead to higher communication costs in the solver (this assumption was also used for the run time modelling in [76]). Overall, this leads to an accumulation of long-running grids in only a few groups, which is the reason for the higher load imbalance.



(a) ...with the METIS-optimized assignment, cf. Section 4.6.2



(b) ...with round-robin assignment (and balanced DOF for the lower diagonals)

Figure 5.24.: The scenario ($n_g = 64, n_p = 256$) run on SuperMUC-NG with different assignments of component grids to process groups. The bars denote the mean times required for different parts of Algorithm 2.1, the error bars denote the respective standard deviations.

Of course, this change also affects the communication volume of the combination step: For the naive assignment, the size of the sparse grid that needs to be allocated increases, from formerly 0.99 GB per core to 1.46 GB per core. In the ChunkedOutgroupSGReduce variant with a chunk size of 64 MiB per core, this corresponds to 22 chunks (and consecutive calls to `MPI_Allreduce`) as opposed to 15 chunks for the METIS-optimized assignment. The overall time required for ChunkedOutgroupSGReduce is labelled 'Reduce/Scatter' in Figure 5.24, and the time required changes only within the standard deviation—which is less than anticipated.

Conclusively, since the transfer between the groups is relatively cheap at this level of parallelization, the naive assignment is preferable for balancing the load in the (significantly more expensive) solver update step. This naive assignment effectively transforms the ChunkedOutgroupSGReduce algorithm into a chunked sparse grid reduce algorithm, since the main diagonal of the scheme is distributed in round-robin manner, and it becomes impossible for one group to get assigned an entire patch of neighboring grids in the main diagonal (a $d-1$ dimensional simplex).

Of course, the situation is going to be different when the combination takes places across low-bandwidth networks where communication volumes are very important, such as for the widely-distributed setup in Section 5.5.

5.4.4. I/O Timings for File-Based Combination

As previously mentioned, one of the goals of the experiments was the evaluation of the I/O timings on the different systems for the different degrees of parallelization.

Figures 5.25 to 5.28 show the mean timings required to write and read the conjoint sparse grid file as part of the file-based combination. Recall that the conjoint file size is going to depend linearly on the process group size n_p, with sizes ranging from about one GiB to two TiB. Fritz and HAWK operate on Lustre file systems, while the other two systems use the proprietary IBM GPFS, compare Table 5.1.

The most striking fact about the measurements can be seen from the value ranges, which are adjusted to each plot: Writing and reading on HAWK can take a lot longer than on the other systems. Admittedly, the data points in front of the plots mean an I/O volume of ≈ 2 TiB for HAWK, while it is 1 TiB for SuperMUC-NG and only 0.25 TiB for Fritz and JUWELS. On the other hand, one could argue that file systems on larger machines should correspondingly have higher parallel performance. In particular, even for these data points where we know all systems' timings, such as $(n_g = 1, n_p = 64)$ ranks, the difference is obvious.

Like mentioned in Section 4.5.3, the placement of I/O ranks was specifically optimized for HAWK, where the link from the compute racks to the file system is a known bottleneck. Potentially, the effect of 'spreading out' the I/O ranks can be observed in the HAWK reading times for the larger process groups, which almost linearly decrease for more than two groups, as more and more racks / file system connections are used. Despite this optimization, the achieved read and write rates are not satisfactory for HAWK in comparison to the other machines, which all operate on relatively similar value ranges.

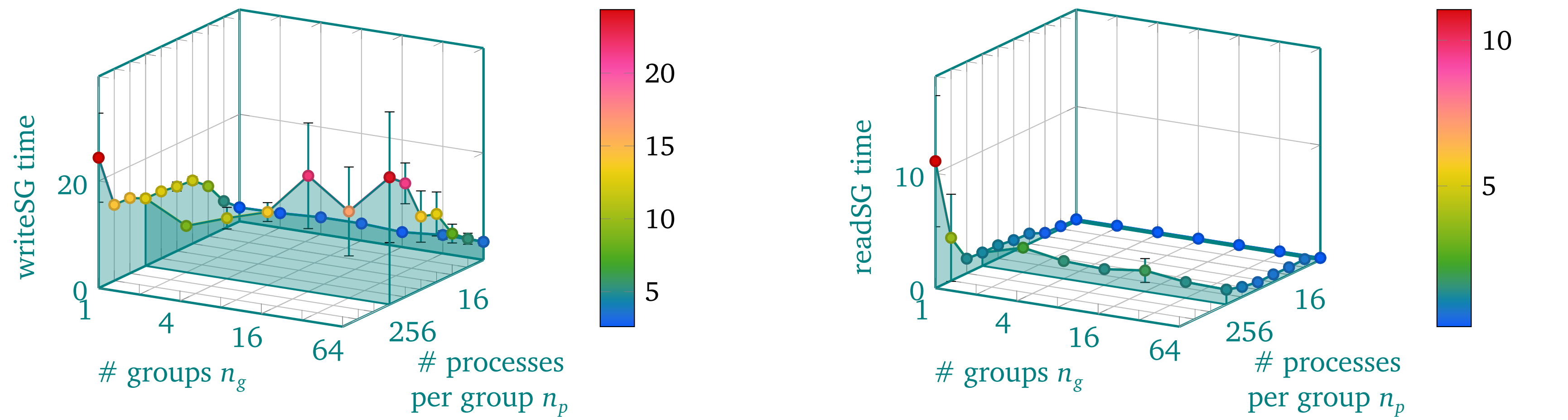


Figure 5.25.: I/O timings on Fritz: Mean times with standard deviations across combination steps and I/O ranks.

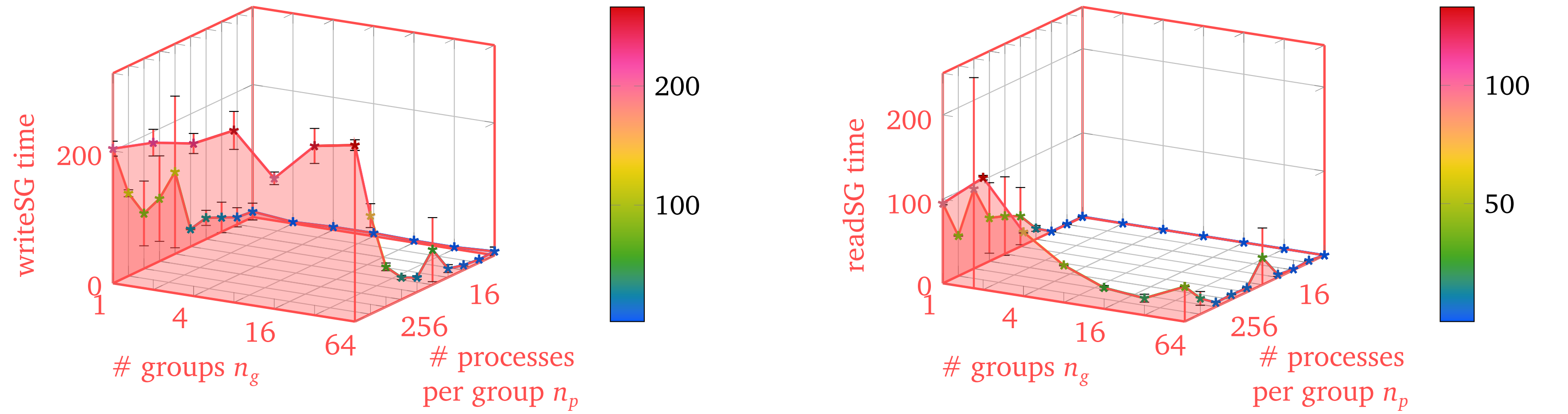


Figure 5.26.: I/O timings on HAWK: Mean times with standard deviations across combination steps and I/O ranks.

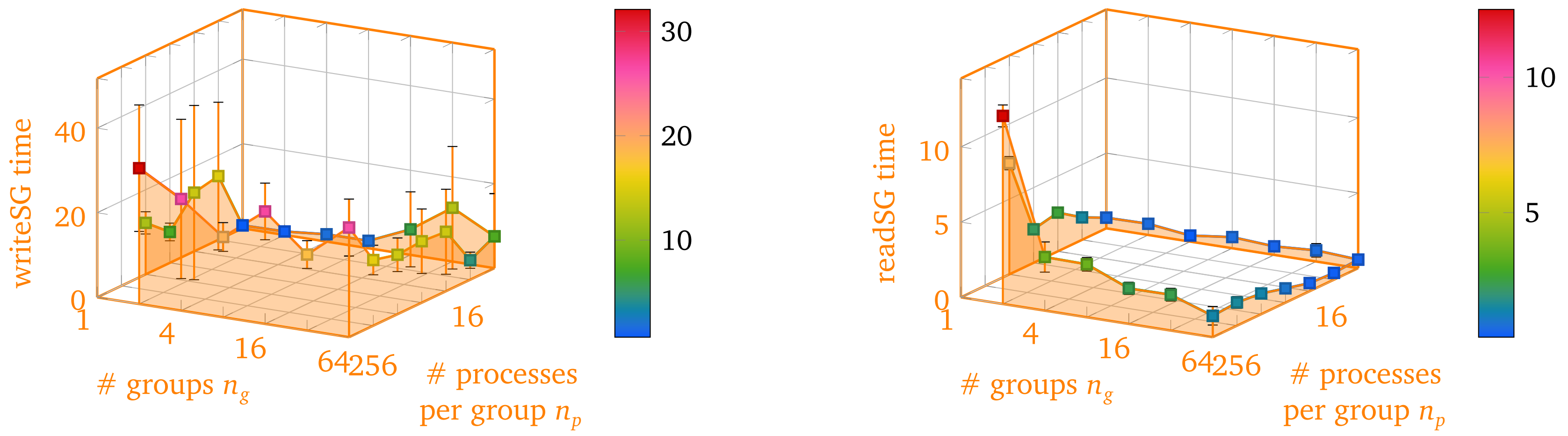


Figure 5.27.: I/O timings on JUWELS: Mean times with standard deviations across combination steps and I/O ranks.

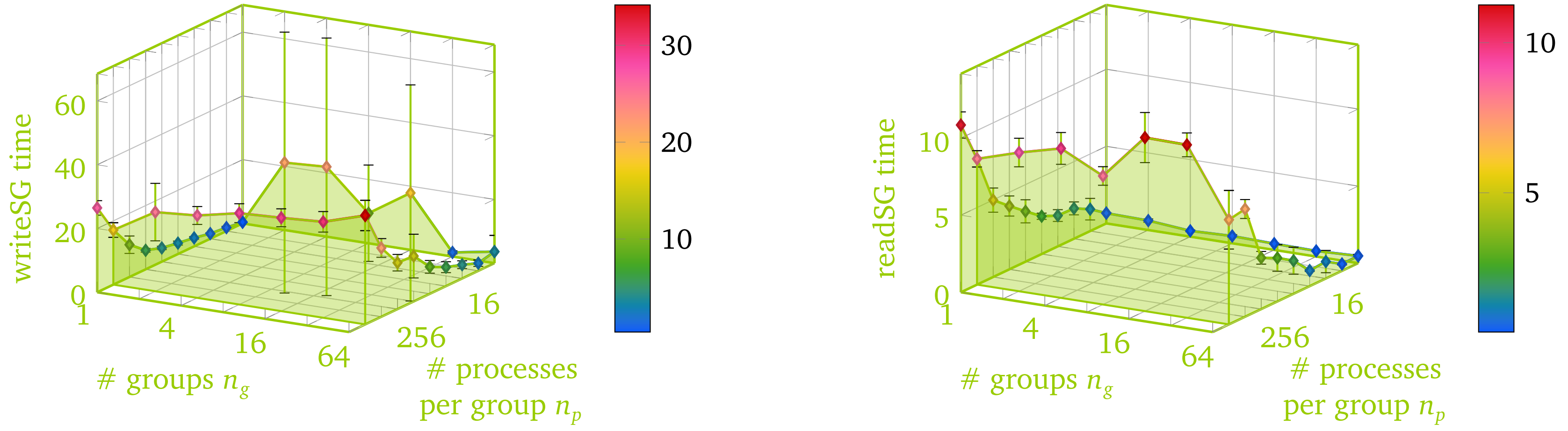


Figure 5.28.: I/O timings on SuperMUC-NG: Mean times with standard deviations across combination steps and I/O ranks.

5.5. Towards Widely-Distributed Simulations at Extreme Scales

Section 4.5 discussed the possibility of connecting two HPC systems to run a widely-distributed simulation. In [138], the author of this thesis and co-authors presented the results of distributing a—comparably small—combination scheme between the two systems HAWK and SuperMUC-NG. The scheme was designed to yield (relatively few) component grids of large size:

Combination scheme for widely distributed [138], 8 groups of $n_p = 8192$			
$\vec{\ell}^{\min}$	$(4,4,4,3,3,3)$	# grids	2975
$\vec{\ell}^{\max}$	$(12,12,12,11,11,11)$	# finest grids	1287
total FG #DOF	9.88×10^{11}	mem. finest grids	4 GiB
total FG memory	7.19 TiB	#DOF FG at $\vec{\ell}^{\max}$	5.90×10^{20}

The paper showcases that the distribution not only works, but also that tolerable transfer rates are going to be achievable for large-scale scenarios. Within a system, sparse grid reduce with the static balanced round-robin approach was used to assign component grids to process groups. The simulations were performed using only MPI, i. e., without hybrid OpenMP parallelization. The two approaches of splitting the combination scheme between the systems discussed in Section 4.6 were evaluated and found to give very similar results for the symmetric case, and only the METIS-based splitting can be used for the asymmetric case, which results in lower transfer volumes. The bandwidth and stability of the (UFTP or other file exchange) connection were identified as pivotal to the performance of the widely-distributed simulation. Also, it became apparent that a good load balance between the systems has to be achieved, and that transfer speeds have to be optimized, to avoid unnecessary idle times.

However, since HAWK was found to lack in I/O performance for large file sizes and high process counts, cf. Section 5.4.4, the results of the experiments between HAWK and SuperMUC-NG are not presented here in detail, but only in comparison to a more recently run setup, and the interested reader is referred to the original publication [138].

This section explores the possibility of connecting a different pair of systems, SuperMUC-NG and JUWELS, to perform a widely-distributed simulation. Since the two systems have a very similar processor architecture and the same file system type, load balance should be easier to achieve. Also, the OpenMP hybrid parallelism is employed, with $n_t = 4$ cores per rank, to save on MPI memory overheads, which otherwise were especially limiting on the SuperMUC-NG system. Since the size of the systems is rather unsymmetrical—SuperMUC-NG has approximately three times as many cores/nodes/GB of main memory as JUWELS—the METIS-based assignment of the component grids to the systems is used again.

The *target scenario* for groups of 512 ranks or 2048 cores

Combination scheme for widely distributed, 163 groups of $n_p = 512$

$\vec{\ell}^{\min}$	$(3,2,2,2,2,2)$	# grids	88,571
$\vec{\ell}^{\max}$	$(20,19,19,19,19,19)$	# finest grids	26,334
total FG #DOF	4.57×10^{13}	mem. finest grids	8 GiB
total FG memory	332.55 TiB	#DOF FG at $\vec{\ell}^{\max}$	4.15×10^{34}

requires 163 process groups, if we want to approach the (experimentally validated) limit of fitting ≈ 1.1 GB of component grid data per core. Of these 163 groups, a maximum of 148 can be placed on SuperMUC-NG, and up to 53 on JUWELS, when the systems are used almost entirely at the same time. By assigning the full 148 groups to SuperMUC-NG and the remaining 15 groups to JUWELS, a conjoint sparse grid transfer volume of only 612 GB can be achieved.

Recall that when using METIS-based assignments of grids to groups, only little effect was found for the run time of the combination step as opposed to random assignment, cf. Section 5.4.3. This is different here, as the limited bandwidth of the widely-distributed combination calls for reduced combination volumes. With the naive assignment of the scheme's component grids to HPC systems, the transfer volume would be 6.82 TiB for the target scenario, which would translate to more than 12 times higher UFTP transfer times. This illustrates the necessity of optimizing the splitting of the combination scheme $\mathcal{I}^{\mathrm{CT}}$ between the systems.

To summarize the data sizes: For running the entire target scenario, a total of 332.55 TiB of full grid data is required to even store the necessary data for the six-dimensional unknown field u^{CT}. Of this data, only 612 GB is required to be transferred between the systems as conjoint data in a widely-distributed simulation. If instead, one wanted to perform a monolithic simulation at the maximum resolution level of $\vec{\ell}^{\max}$, one would have to work on $256 \cdot 2^{110}$B, or 332,307 *quettabyte* which is the largest currently defined binary unit. This size is too much to be stored on any media currently available and by far too much for the main memory of all the Top500 [153] combined.

To prepare DisCoTec for such extreme simulation scales, one fifteenth of the scheme was run on 10th August 2023, with ten process groups on SuperMUC-NG and one process group on JUWELS, and the results are evaluated in the remainder of the section. Although only a part of the simulation was run, the conjoint sparse grid was written out at the full file size of 612 GB to test the performance of the I/O operations and the UFTP transfer, in particular for potentially concurrent accesses to the GPFS file systems. Conclusively, the transfer timings can be considered representative of the target scenario; Furthermore, the feasibility of scaling up to the full system sizes of both machines was already demonstrated in Section 5.4.2.

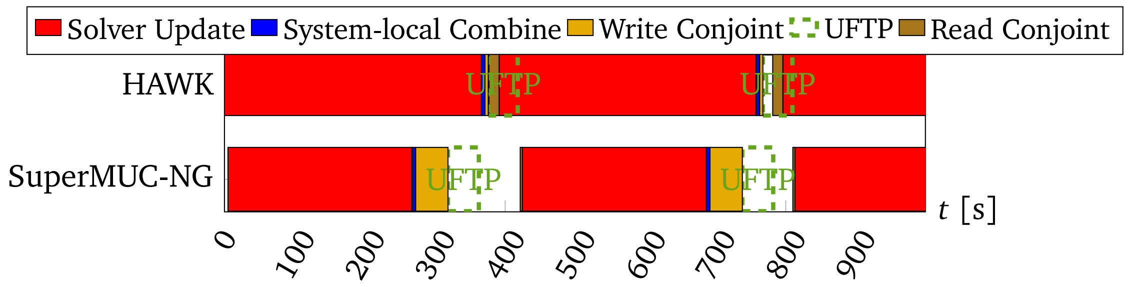
Overheads of Widely Distributed Weak Scaling

This paragraph analyzes the results of the new scenario in the light of the previous results, and assesses the changes and likely causes.

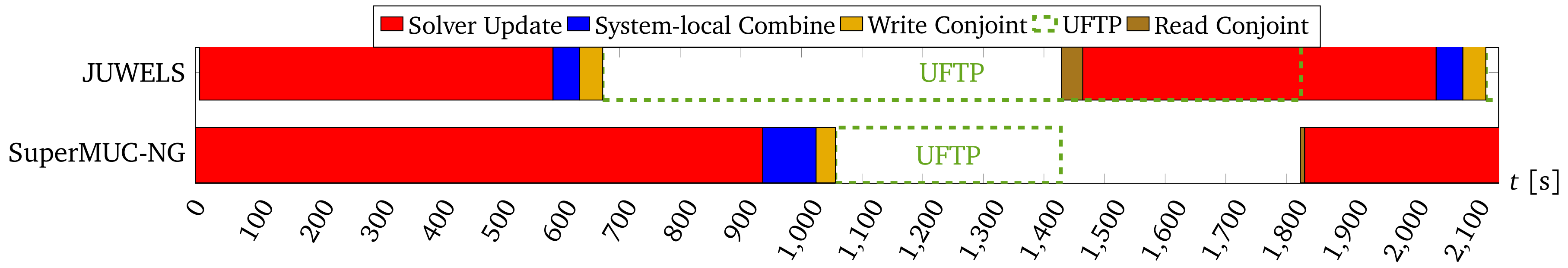
For the previous experiment [138], the DisCoTec simulations on HAWK and SuperMUC-NG were run on groups of 8192 ranks (equal to cores, since OpenMP was not used). Up to eight groups were used, with proportionally sized partitions of the combination scheme. As the new experiment used ChunkedOutgroupSGReduce, more main memory per core could be used for the component grid data—independent of the group size, which was set to 512 ranks (or 2048 cores) per group. Ten groups were used for SuperMUC-NG and one group for JUWELS. The difference in the component grid memory per core—1047 MiB per core as opposed to previously 115 MiB per core—meant that many more tasks could be assigned to a single group: Up to 588 tasks were now assigned per group (out of 81680 in the target scenario) as opposed to previously up to 376 tasks per group. At the same time, the grid part held by each core would be eight times as large (factor 2 from the combination scheme itself, factor 4 from the smaller group size). To account for this, the number of advection solver time steps per combination step was reduced from 300 to 36.

For both experiments, the transfers were triggered from the UFTP client programs installed on SuperMUC-NG's gateway nodes. The size of the conjoint sparse grid files was 76 GB and 612 GB, respectively.

Figure 5.29 shows the time line of the different parts in the resulting algorithm.



(a) Symmetrical partitioning of the combination scheme between HAWK and SuperMUC-NG, with 4 groups on SuperMUC-NG and 4 groups on HAWK.



(b) First two (and only) solver steps in the widely-distributed scenario between SuperMUC-NG and JUWELS. The bar lengths are obtained by extracting the maximum time taken by any rank.

Figure 5.29.: Timelines of the first few time steps in the widely-distributed simulations.

If one wants to directly compare the widely-distributed to the system-local setting, anything that is not part of the 'Solver Update' or 'System-local Combine' steps (the red and blue fields in the figure) can be considered an overhead of the widely-distributed approach. In particular, these are the times required for writing the conjoint sparse grid, and for waiting for the UFTP-transferred file to arrive, for reading, and for reducing the received with the system-local data. When comparing the two scenarios, it is important to keep in mind that the newer simulation operated on approximately nine times as many DOF per core, and the UFTP transfer volume was approximately eight times as high.

As one can see from Figure 5.29b, the challenges with the new scenario are the same ones as already identified in [138]: load imbalance between the systems and low transfer rates.

For the recent experiment, the load imbalance between the system can be explained by load imbalance *within* the SuperMUC-NG system: Unfortunately, the imbalance introduced by the METIS-based assignment on the process-group level, cf. Section 5.4.3, had not yet been identified and was also used in this simulation. While the maximum solver update time on SuperMUC-NG was 936 s—resulting in a load imbalance of ≈ 1.55—the mean time was only 605 second, very similar to JUWELS' mean time of 582 on SuperMUC-NG. As discussed in Section 5.4.3, this issue can be resolved (in a statistical sense) by a more naive assignment of tasks to process groups. This had in fact been used for the previous experiment, which led to negligible load imbalance within the systems.

The timings in Figure 5.29b can be used to derive an upper-bound estimate on the DisCoTec timings for the whole target scenario: When using the naive assignment, the solver update time should be close to the mean time of 605 s on SuperMUC-NG for both machines, as this part is going to stay embarrassingly parallel as more groups are added. (The mean time on SuperMUC-NG can be considered more expressive than the JUWELS result, which was not averaged over several process groups and therefore likely has a higher bias). The same considerations hold for the basis change operations with 12 s for the hierarchization and 21 s for the dehierarchization.

The remaining part of the system-local combination step, the reduce and scatter operations, took 32 s on SuperMUC-NG, and can be expected to take four times as long per chunk for the full target scheme, as the number of groups is increased by a factor of ≈ 15 and the scaling in the chunked reduction operation can be expected to be approximately logarithmic. However, there is a catch: By considering the full combination scheme with round-robin assignment, the number of chunks that need to be communicated is going to significantly increase—after all, the current measurements were optimized to keep as many subspaces as possible ‘ingroup’, which will necessarily become ‘outgroup’ through the naive assignment. Conclusively, the current number—7 chunks per combination step—can get as high as 52 chunks per combination step. The effects of this are hard to predict, since the horizontal reduction patterns may lead to a certain degree of balancing on the interconnect, which could alleviate the increase in communication volume. Still, an upper-bound estimate would be a factor of ≈ 7.5 increase in the time required for the reduce and scatter operations. Considering the I/O figures for SuperMUC-NG and JUWELS in Section 5.4.4, one can assume that the write and read times for a file of the same size and groups of the same size will be in the same order of magnitude, with a potential for decreasing read times for more than one group on JUWELS. Overall, this leads to a (DisCoTec-only) time of 27 min per time step in the worst case for the target scenario—and likely less—where at least 37% of the time can be spent on the solver update itself.

The latencies induced by the UFTP transfers are a more difficult problem to address. Particularly for transfers going out of JUWELS, high variances were observed, with some UFTP threads dragging the entire file transfer. These are at least partly amplified by the underlying TCP protocol: As soon as any packet loss occurs, the transfer rate is reduced significantly, and the respective thread can take a long time (several minutes) to recover its initial bandwidth. Additional challenges are posed by the shared nature of the media, in particular the UFTP client and server nodes which can be used by other users at the same time.

Of course, there are potential technical solutions to this, namely

1. *Compression* of the conjoint sparse grid file before transfer could reduce the transfer volume and thus the transfer time.

2. GridFTP would be another file exchange tool which can use the UDT protocol in place of TCP, where packet loss should not lead to the same massive bandwidth cuts.

3. Investigating the firewall and network settings and optimizing the transfer trigger sites accordingly may also help to improve the transfer rates.

Furthermore, at an organizational level, the prioritization of production transfers over storage transfers could be an answer to the issue of low transfer rates resulting from competing transfers on the same infrastructure. Conclusively, faster transfers are possible, and the timings plotted in Figure 5.29 should be considered mere snapshots of the current state of widely-distributed simulations.

These measurements allow for a positive assessment of future widely-distributed simulations at scale.

Chapter 6

Conclusion and Outlook: Towards Exascale Computations for Net-Positive Fusion Energy

Although a great tool for plasma fusion science, the numerical simulation of confined plasmas poses a challenge. This is an undisputed fact in both the controlled fusion research [20, 107] and the HPC [35] communities. The reason for this is the Curse of Dimensionality, which is a result of the high dimensionality of the Vlasov equation and the fine scales that need to be resolved for high-fidelity plasma simulations.

The sparse grid combination technique is one potential solution to this problem: It adds a multiscale approach with embarrassing parallelism to existing grid-based solvers. The CT can achieve similar accuracy as the solver when run on the full grid at a fraction of the cost, in both compute time and main memory. This holds especially true for higher-dimensional problems, of which the Vlasov equation is one example.

Some numerical and computational aspects in the sparse grid combination technique were further developed as part of this work, and the results are promising: The accuracy was increased and numerical stability as well as conservation of mass were introduced to turbulent plasma simulations with the CT. Due to the loose coupling, the CT allows one to scale up to full HPC systems, and even beyond that, to two HPC systems. At the same time, however, there is a plethora of scientific and technical questions that remain to be answered.

This chapter looks at both the contributions of and the open challenges arising from this work.

6.1. Stable and Moment-Conserving Simulations at the Memory Limit of More than Entire HPC Systems

Chapter 2 introduced the hierarchical hat basis from the perspective of biorthogonal wavelets, which is relatively uncommon in the sparse grid literature. However, this viewpoint helps to understand the nodal spaces and increment spaces of the multiscale construction in a more generalized way: In wavelet terms, the nodal functions become scaling functions, hierarchical increments become wavelet coefficients etc. And, importantly, the hat function can be replaced for instance by mass conserving basis functions. This trades the hierarchical hat function's interpolating property for other desirable properties: accuracy, conservation, and stability of the basis functions. Based on these multiscale functions, one can construct sparse grid spaces, which can very efficiently represent higher-dimensional functions by allowing for very anisotropic function spaces, while omitting function spaces that are very finely resolved in each dimension. The sparse grid combination technique is introduced as a method that can bridge the gap between the 'ordinary' world of full grid spaces and the multiscale world of sparse grid spaces. Very closely connected to Smolyak quadrature, it achieves this by combining multiple coarsely resolved full grids—the component grids—into a sparse grid with a combination formula derived for mutual error cancellation of the full grids. For PDE solvers, the time-stepping CT is of particular interest.

Chapter 3 shows the effects of using the mass-conserving basis functions for the CT in the context of three different setups: Some simple two-dimensional examples that illustrate mass loss and instability with the hat basis, the advection equation in two to six dimensions (where the accuracy of the results improves significantly) and the Vlasov solver SeLaLib. The SeLaLib simulations in particular showcase that the numerical instabilities encountered with Vlasov solvers in the CT can be overcome by using mass-conserving basis functions. Although the resolutions in the combination schemes for these simulations were not particularly adapted to the specific problem, the accuracy of the results is already comparable to the isotropic full grid solution at the same memory footprint.

The algorithms and code enabling the simulations in this thesis are presented in Chapter 4: The DisCoTec framework adds another level of (distributed-memory) parallelism on top of existing solvers' parallelism, which is particularly useful for achieving scalability when considering dataflow. Over the course of the presented research, DisCoTec was open-sourced and further developed. The most notable additions are the mass-conserving basis transforms, three additional distributed reduction operators, optimizations for low memory overhead at scale, and highly efficient parallel sparse grid I/O, which can be used for widely-distributed simulations that synchronously span more than one HPC systems.

Chapter 5 assesses DisCoTec's scalability. The gyrokinetic code GENE was identified to not be the best choice for the CT with DisCoTec, due to the very high memory requirements of only a few of the GENE instances. However, the lessons learned from GENE helped to design comprehensive strong and weak scaling scenarios for DisCoTec applied to the six-dimensional advection problem. A variety of new features were used to scale the program up to four full German supercomputing systems. In particular, weak scaling showcased that this can be achieved while scratching the memory limits of the supercomputers, and using more than half of the available memory for the full grid simulation instances. And even beyond full systems, DisCoTec was put to practical use in performing synchronous simulations on two sets of two supercomputers together (the widely-distributed CT). This approach is particularly interesting for federated infrastructure such as the German national Tier-0/1 systems, since only a small fraction of simulation data needs to be exchanged between systems. Two algorithms to derive especially communication-avoiding partitionings of the combination schemes were presented as part of this work in Section 4.6. Pairings of the HAWK and SuperMUC-NG as well as the JUWELS and SuperMUC-NG systems were used to perform these simulations and to extrapolate run times for extreme scale scenarios surpassing the memory available on a single system.

Accordingly, the challenges of conservation and stability as well as scalability up to and beyond full system sizes at the memory limit were addressed in this work.

6.2. Future Research Directions

Some insights in this work were eased through the realization that the hierarchical hat functions are only one possible basis function to construct a sparse grid space, and that all compactly supported wavelets may be used to construct sparse grids. To allow this, one has to accept that the resulting transformations do not necessarily have the interpolating property, which e. g., allows one to look at piecewise polynomial functions as sparse grid spaces through Alpert multiwavelets [3, 157]. Further exploring possible multiscale functions for sparse grids and the CT, starting with the original CDF wavelets [25] and (possibly problem-tailored) lifting wavelets [154] as well as Alpert multiwavelets, is a very promising direction for future research. An example of a problem-tailored transformation could be the adaptation of the mass-conserving basis functions to the gyrokinetic coordinate system, to conserve the mass in gyrokinetic solvers used with the CT. Also, the full-weighting basis function hints at how intricately biorthogonal wavelets and well-known geometric multigrid methods could be connected. Using wavelet transforms for multigrid restriction and prolongation operators could lead to improved stability, accuracy, and conservation for these methods as well.

Another important aspect that could not be explored in this work is adaptation in the time-stepping CT. Particularly for Vlasov solvers, dynamic adaptation of the grid discretization (based on spatially-adaptive CT with block-structured grids) could give a vital boost in simulation accuracy for a given amount of compute time and memory. Considering the error characteristics of the CT, it could also make sense to detect how much the advection directions in the simulation change, and to adapt the combination frequency accordingly. This would mean that during the initial linear phase of a turbulence simulation, only few combinations would have to be performed, while during the turbulent nonlinear phase, more combinations would be performed, saving time on these parts of the simulation that tend to be of low interest to domain scientists.

Developing the necessary data structures for arbitrary wavelets and adaptivity in a distributed setting poses a challenge, yet at the same time, once such a piece of code exists it would have a great potential to boost plasma simulation research. Quite unfortunately, great open source tools this implementation could build upon, such as preCICE [22] and p4est [18], are currently limited to only two or three spatial dimensions. At the same time, some interesting advances on adaptive domain decompositions for the CT are already being developed [62].

A promising recent research direction, which evolved parallel in time to the work presented in this thesis, is spanned by low-rank methods [2, 40, 41]. Low-rank methods decompose the distribution function as well as the operations into low-rank representations for the space and velocity dimensions, respectively. It would be interesting to investigate the use of hierarchical basis constructions in the low-rank discretizations, which could lead to straightforward sparsification and adaptivity in the low-rank setting.

The approaches presented in the thesis can (and should) be applied to other Vlasov solvers, such as Gysela [57, 58], hyper.deal [114], or SLDG [39, 106], the latter two of which use Discontinuous Galerkin methods and could make use of

Alpert multiwavelets as multiscale functions. To use existing codes with DisCoTec, it needs to be made sure that the main memory scaling of the solver is not too anisotropic—as was the case with GENE, cf. Section 5.3.2.

Lastly, the widely-distributed CT is a tool that is now available to perform Vlasov simulations at scales that are larger than the memory of a single supercomputer, to achieve resolutions that are currently unheard of.

Reproducibility References

The full specifications and outputs for the results presented in Sections 3.4 and 3.5.1 are published in [130].

The GENE experiments described in Section 5.3 were conducted on DisCoTec`57dceaa2` with GENE`d19569eb` (to be found in fork `https://gitlab.mpcdf.mpg.de/g-michaelobersteiner/gene-dev.git`). The scripts to generate these scenarios along with the GENE parameter specification can be found in [129], in the folder `gene_distributed_scaling`. Some of the raw output data is also available in this thesis' reproducibility data repository [131].

The advection's strong scaling, evaluated in Section 5.4.1 was run with DisCoTec, commit `ba519e39`, executable `combi_workers_only`. The weak scaling, Section 5.4.2, was run with commit `4b4cea5f`, executable `distributed_third_level_worker_only`. The widely-distributed simulations from Section 5.5 were performed with commit `385e7230` of DisCoTec, executable `distributed_third_level_worker_only`. The assignments of component grids to process groups were generated with commit `dcf6d76d` of [128].

The full input and output data for the measurements in Sections 5.4 and 5.5 is available in the data repository [131] for reproducibility.

The data to reproduce the scaling measurements for DisCoTec + SeLaLib presented in Appendix A.1 is available in [133].

Acknowledgements

This work was supported by the German Research Foundation (DFG) through the Priority Programme 1648 Software for Exascale Computing (SPPEXA).

We acknowledge the support by the Stuttgart Center for Simulation Science (SimTech).

HPC Systems

The authors gratefully acknowledge the scientific support and HPC resources provided by the Erlangen National High Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) under the NHR project a105cb. NHR funding is provided by federal and Bavarian state authorities. NHR@FAU hardware is partially funded by the German Research Foundation (DFG) — 440719683.

The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (`www.gauss-centre.eu`) for funding this project by providing computing time on the GCS Supercomputer HAWK at Höchstleistungsrechenzentrum Stuttgart (`www.hlrs.de`). The simulations were performed on the national supercomputer HPE Apollo Hawk at the High Performance Computing Center Stuttgart (HLRS) under the grant number 42247.

The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (`www.gauss-centre.eu`) for funding this project by providing computing time on the GCS Supercomputer JUWELS at Jülich Supercomputing Centre (JSC).

The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (`www.gauss-centre.eu`) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre (`www.lrz.de`).

Tooling

This thesis was typeset with LaTeX/pdfLaTeX and is based on the excellent scientific thesis template [105]. GitHub Copilot [55] was used for text and LaTeX assistance.

The author would like to highlight some particularly instrumental LaTeX packages for this work: The siunitx package [171] allowed using binary units as well as pseudo-units such as '%' or 'DOF' in a way very suitable for this topic. Most of the graphics are based on Ti*k*Z [156]. In particular, all plots were generated with PGFPLOTS [47]. The plots Figures 3.1 to 3.3, 5.5 to 5.21, 5.23 and 5.25 to 5.28 further employ the perceptually uniform CET-R3 color map [19, 98]. For single images, sources and tools may further be given in the captions.

People

Thanks to Stefan Zimmer, Julian Valentin, Gregor Daiß, Raphael Leiteritz, Marcel Breyer, Alexander Van Craen, Peter Münch, Katharina Kormann, Alejandro Banon Navarro, Felix Wilms, Tilman Dannert, Hans-Joachim Bungartz, Frank Jenko, Michael Griebel, Mikael Simberg, John Biddiscombe, Christoph Niethammer, Philipp Offenhäuser, Michael Obersteiner, and Johannes Rentrop for fruitful discussion and collaboration.

I had great fun co-organizing two SPPEXA women's workshops, which I attribute to my partners in this endeavor, Sarah Huber and Neda EbrahimiPour. I would also like to thank Marcel Hurler, Marvin Dostal, Marius Göhring, Benjamin Castellaz and Maureen Kosiol for doing neat student research under my supervision. Also, thanks to Maureen Kosiol for coining the term 'decombination'.

I am utterly grateful for the support received at the various HPC sites, and I would like to highlight in particular our projects' main contact persons: Martin Bernreuther (HLRS), Martin Ohlerich (LRZ), Bernd Schuller (JSC), and Thomas Zeiser (NHR@FAU).

Special thanks to my advisor Dirk Pflüger, who taught me so many things I did not even know I should know, and who mentored me through the toughest times.

And of course to my family and friends for their kindness and support.

Bibliography

[1] J. M. Alam, N. K. .-. Kevlahan, O. V. Vasilyev. ‘Simultaneous space–time adaptive wavelet solution of nonlinear parabolic differential equations’. In: *Journal of Computational Physics* 214.2 (May 20, 2006), pp. 829–857. URL: `https://www.sciencedirect.com/science/article/pii/S0021999105004754` (visited on 06/16/2023) (cit. on p. 39).

[2] F. Allmann-Rahn, R. Grauer, K. Kormann. ‘A Parallel Low-Rank Solver for the Six-Dimensional Vlasov–Maxwell Equations’. In: *Journal of Computational Physics* 469 (Nov. 15, 2022), p. 111562. URL: `https://www.sciencedirect.com/science/article/pii/S0021999122006246` (visited on 09/25/2023) (cit. on p. 142).

[3] B. K. Alpert. ‘A class of bases in L^2 for the sparse representation of integral operators’. In: *SIAM journal on Mathematical Analysis* 24.1 (1993), pp. 246–262 (cit. on pp. 24, 38, 141).

[4] G. Avila, T. Carrington Jr. ‘A multi-dimensional Smolyak collocation method in curvilinear coordinates for computing vibrational spectra’. In: *The Journal of Chemical Physics* 143.21 (Dec. 2, 2015), p. 214108. URL: `https://doi.org/10.1063/1.4936294` (visited on 05/08/2023) (cit. on p. 25).

[5] A. Baker. *Nuclear Fusion Finally Finds Its Place in the Sun*. Time. Nov. 24, 2021. URL: `https://time.com/6123622/nuclear-fusion-viability/` (visited on 05/02/2023) (cit. on p. 11).

[6] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefler, S. Kumar, E. Lusk, R. Thakur, J. Träff. ‘MPI on Millions of Cores’. In: *Parallel Processing Letters* 21 (Mar. 1, 2011), pp. 45–60 (cit. on p. 77).

[7] S. Bauer, H.-P. Bunge, D. Drzisga, S. Ghelichkhan, M. Huber, N. Kohl, M. Mohr, U. Rüde, D. Thönnes, B. Wohlmuth. ‘TerraNeo—Mantle Convection Beyond a Trillion Degrees of Freedom’. In: *Software for Exascale Computing - SPPEXA 2016-2019*. Ed. by H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, W. E. Nagel. Lecture Notes in Computational Science and Engineering. Cham: Springer International Publishing, 2020, pp. 569–610 (cit. on p. 67).

[8] T. Beisel, E. Gabriel, M. Resch. ‘An extension to MPI for distributed computing on MPPs’. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Ed. by M. Bubak, J. Dongarra, J. Waśniewski. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 75–82 (cit. on p. 68).

[9] T. Ben-Nun, J. de Fine Licht, A. N. Ziogas, T. Schneider, T. Hoefler. ‘Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures’. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’19. New York, NY, USA: Association for Computing Machinery, Nov. 17, 2019, pp. 1–14. URL: `https://dl.acm.org/doi/10.1145/3295500.3356173` (visited on 05/08/2023) (cit. on p. 66).

[10] J. Benk, D. Pflüger. ‘Hybrid parallel solutions of the Black-Scholes PDE with the truncated combination technique’. In: *2012 International Conference on High Performance Computing Simulation (HPCS)*. 2012 International Conference on High Performance Computing Simulation (HPCS). July 2012, pp. 678–683 (cit. on p. 31).

[11] N. Besse, E. Deriaz, É. Madaule. ‘Adaptive multiresolution semi-Lagrangian discontinuous Galerkin methods for the Vlasov equations’. In: *Journal of Computational Physics* 332 (Mar. 1, 2017), pp. 376–417. URL: `https://www.sciencedirect.com/science/article/pii/S0021999116306441` (visited on 06/16/2023) (cit. on p. 39).

[12] N. Besse, G. Latu, A. Ghizzo, E. Sonnendrücker, P. Bertrand. ‘A wavelet-MRA-based adaptive semi-Lagrangian method for the relativistic Vlasov–Maxwell system’. In: *Journal of Computational Physics* 227.16 (Aug. 10, 2008), pp. 7889–7916. URL: `https://www.sciencedirect.com/science/article/pii/S0021999108002672` (visited on 05/13/2022) (cit. on p. 39).

[13] R.-P. Braun, A. Schippel, O. Andryushchenko, J. Weingart, S. Neidlinger, M. Eiselt, M. Alfiad, T. William, S. Höhlig, N. Schäfer, S. Tibuleac, W. Weiershausen, E. Beier. ‘Tbit/s 1000 Km field trial, achieving increased spectral efficiency, SDN enabled application traffic, and passive wavelength switching’. In: *2015 Opto-Electronics and Communications Conference (OECC)*. 2015 Opto-Electronics and Communications Conference (OECC). ISSN: 2166-8892. June 2015, pp. 1–3 (cit. on p. 68).

[14] A. J. Brizard, T. S. Hahm. ‘Foundations of Nonlinear Gyrokinetic Theory’. In: *Reviews of modern physics* 79.2 (2007), p. 421 (cit. on pp. 14, 98).

[15] H.-J. Bungartz, M. Griebel. ‘Sparse grids’. In: *Acta Numerica* 13 (May 2004). Publisher: Cambridge University Press, pp. 147–269. URL: `https://www.cambridge.org/core/journals/acta-numerica/article/sparse-grids/47EA2993DB84C9D231BB96ECB26F615C` (visited on 05/04/2023) (cit. on pp. 21, 25, 27, 29).

[16] H.-J. Bungartz, P. Neumann, W. E. Nagel. *Software for Exascale Computing - SPPEXA 2013-2015*. 1st ed. Springer Publishing Company, Incorporated, May 2018. 565 pp. (cit. on p. 65).

[17] H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, W. E. Nagel, eds. *Software for Exascale Computing - SPPEXA 2016-2019*. Springer Nature, 2020. URL: `https://library.oapen.org/handle/20.500.12657/41289` (visited on 01/19/2022) (cit. on p. 65).

[18] C. Burstedde, L. C. Wilcox, O. Ghattas. 'p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees'. In: *SIAM Journal on Scientific Computing* 33.3 (2011), pp. 1103–1133 (cit. on p. 142).

[19] *CET Perceptually Uniform Colour Maps*. URL: `https://colorcet.com/index.html` (visited on 08/09/2023) (cit. on p. 146).

[20] C.-S. Chang, M. Greenwald, K. Riley, K. Antypas, R. Coffey, E. Dart, S. Dosanjh, R. Gerber, J. Hack, I. Monga, M. E. Papka, L. Rotman, T. Straatsma, J. Wells, R. Andre, D. Bernholdt, A. Bhattacharjee, P. Bonoli, I. Boyd, S. Bulanov, J. R. Cary, Y. Chen, D. Curreli, D. R. Ernst, S. Ethier, D. Green, R. Hager, A. Hakim, A. Hassanein, D. Hatch, E. D. Held, N. Howard, V. A. Izzo, S. Jardin, T. G. Jenkins, F. Jenko, A. Kemp, J. King, A. Kritz, P. Krstic, S. E. Kruger, R. Kurtz, Z. Lin, B. Loring, G. Nandipati, A. Y. Pankin, S. Parker, D. Perez, A. Y. Pigarov, F. Poli, M. J. Pueschel, T. Rafiq, O. Rübel, W. Setyawan, V. A. Sizyuk, D. N. Smithe, C. R. Sovinec, M. Turner, M. Umansky, J.-L. Vay, J. Verboncoeur, H. Vincenti, A. Voter, W. Wang, B. Wirth, J. Wright, X. Yuan. *Fusion Energy Sciences Exascale Requirements Review. An Office of Science review sponsored jointly by Advanced Scientific Computing Research and Fusion Energy Sciences, January 27-29, 2016, Gaithersburg, Maryland*. US-DOE Office of Science (SC), Washington, DC (United States). Offices of Advanced Scientific Computing Research and Fusion Energy Sciences, Feb. 1, 2017. URL: `https://www.osti.gov/biblio/1375639` (visited on 05/08/2023) (cit. on pp. 15, 65, 139).

[21] C. Z. Cheng, G. Knorr. 'The integration of the Vlasov equation in configuration space'. In: *Journal of Computational Physics* 22.3 (Nov. 1, 1976), pp. 330–351. URL: `http://www.sciencedirect.com/science/article/pii/002199917690053X` (visited on 08/14/2019) (cit. on pp. 54, 106).

[22] G. Chourdakis, K. Davis, B. Rodenberg, M. Schulte, F. Simonis, B. Uekermann, G. Abrams, H. Bungartz, L. Cheung Yau, I. Desai, K. Eder, R. Hertrich, F. Lindner, A. Rusch, D. Sashko, D. Schneider, A. Totounferoush, D. Volland, P. Vollmer,

O. Koseomur. ‘preCICE v2: A sustainable and user-friendly coupling library [version 2; peer review: 2 approved]’. In: *Open Research Europe* 2.51 (2022). URL: `https://doi.org/10.12688/openreseurope.14445.2` (cit. on p. 142).

[23] C. E. Clark. ‘Renewable energy R&D funding history: A comparison with funding for nuclear energy, fossil energy, energy efficiency, and electric systems R&D’. In: *Congressional Research Service. Source:* `https://sgp.fas.org/crs/misc/{RS}22858.pdf` (2018) (cit. on p. 11).

[24] L. Coblentz. *29 th ITER Council: Steady progress despite challenges including COVID-19*. Nov. 18, 2021. URL: `https://www.iter.org/doc/www/content/com/Lists/list_items/Attachments/972/2021_11_IC-29.pdf` (visited on 04/01/2022) (cit. on p. 11).

[25] A. Cohen, I. Daubechies, J.-C. Feauveau. ‘Biorthogonal bases of compactly supported wavelets’. In: *Communications on Pure and Applied Mathematics* 45.5 (1992), pp. 485–560. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160450502` (visited on 12/08/2021) (cit. on pp. 23–27, 141).

[26] A. Cohen. *Numerical Analysis of Wavelet Methods*. ISSN. Elsevier Science, 2003. URL: `https://books.google.at/books?id=Dz9RnDItrAYC` (cit. on p. 27).

[27] *Configuration — JUWELS User Documentation Documentation*. URL: `https://apps.fz-juelich.de/jsc/hps/juwels/configuration.html` (visited on 06/30/2023) (cit. on pp. 92, 93).

[28] R. Courant, K. O. Friedrichs, H. Lewy. ‘Über die partiellen Differenzengleichungen der mathematischen Physik’. In: *Mathematische Annalen* 100.1 (Dec. 1, 1928), pp. 32–74. URL: `https://doi.org/10.1007/BF01448839` (visited on 08/30/2023) (cit. on p. 48).

[29] G. Daiß, M. Simberg, A. Reverdell, J. Biddiscombe, T. Pollinger, H. Kaiser, D. Pflüger. ‘Beyond Fork-Join: Integration of Performance Portable Kokkos Kernels with HPX’. In: *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). June 2021, pp. 377–386 (cit. on p. 17).

[30] I. Daubechies. ‘Orthonormal bases of compactly supported wavelets’. In: *Communications on Pure and Applied Mathematics* 41.7 (1988), pp. 909–996. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160410705` (cit. on p. 22).

[31] F. Deluzet, G. Fubiani, L. Garrigues, C. Guillet, J. Narski. 'Sparse Grid Reconstructions for Particle-In-Cell Methods'. In: *ESAIM: Mathematical Modelling and Numerical Analysis* 56.5 (5 Sept. 1, 2022), pp. 1809–1841. URL: https://www.esaim-m2an.org/articles/m2an/abs/2022/05/m2an220004/m2an220004.html (visited on 07/05/2023) (cit. on p. 67).

[32] E. Deriaz, S. Peirani. 'Six-Dimensional Adaptive Simulation of the Vlasov Equations Using a Hierarchical Basis'. In: *Multiscale Modeling & Simulation* 16.2 (Jan. 2018). Publisher: Society for Industrial and Applied Mathematics, pp. 583–614. URL: https://epubs.siam.org/doi/abs/10.1137/16M1108649 (visited on 05/18/2022) (cit. on p. 38).

[33] G. Deslauriers, S. Dubuc. 'Symmetric Iterative Interpolation Processes'. In: *Constructive Approximation: Special Issue: Fractal Approximation*. Ed. by R. A. DeVore, E. B. Saff. Constructive Approximation. Boston, MA: Springer US, 1989, pp. 49–68. URL: https://doi.org/10.1007/978-1-4899-6886-9_3 (visited on 03/09/2022) (cit. on p. 25).

[34] B. Dick, T. Bönisch. 'The AMD EPYC Rome Processor' (HLRS). archived in https://web.archive.org/web/20221226131932/https://kb.hlrs.de/platforms/upload/Processor.pdf. May 3, 2020. URL: https://kb.hlrs.de/platforms/upload/Processor.pdf (cit. on pp. 93, 106).

[35] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, X. Chi, A. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, A. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Z. Jin, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. S. Mueller, W. E. Nagel, H. Nakashima, M. E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Valero, A. van der Steen, J. Vetter, P. Williams, R. Wisniewski, K. Yelick. 'The International Exascale Software Project Roadmap'. In: *The International Journal of High Performance Computing Applications* 25.1 (Feb. 1, 2011), pp. 3–60. URL: https://doi.org/10.1177/1094342010391989 (visited on 05/08/2023) (cit. on pp. 65, 139).

[36] M. Dostal. 'Lastbalancierung durch dynamische Aufgaben-Umverteilung mit der Dünngitter-Kombinationstechnik'. Publisher: Universität Stuttgart. BSc Thesis. 2020. URL: http://elib.uni-stuttgart.de/handle/11682/10956 (cit. on p. 88).

[37] S. Dowling. *Could this be the first nuclear-powered airliner?* July 14, 2016. URL: https://www.bbc.com/future/article/20160713-could-this-be-the-first-nuclear-powered-airliner (visited on 05/02/2023) (cit. on p. 11).

[38] T. Dubos, N. K.-R. Kevlahan. 'A conservative adaptive wavelet method for the shallow-water equations on staggered grids'. In: *Quarterly Journal of the Royal Meteorological Society* 139.677 (2013), pp. 1997–2020. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/qj.2097 (visited on 06/15/2023) (cit. on p. 39).

[39] L. Einkemmer. 'A Performance Comparison of Semi-Lagrangian Discontinuous Galerkin and Spline Based Vlasov Solvers in Four Dimensions'. In: *Journal of Computational Physics* 376 (Jan. 1, 2019), pp. 937–951. URL: https://www.sciencedirect.com/science/article/pii/S0021999118306697 (visited on 08/31/2022) (cit. on p. 142).

[40] L. Einkemmer, I. Joseph. 'A Mass, Momentum, and Energy Conservative Dynamical Low-Rank Scheme for the Vlasov Equation'. In: *Journal of Computational Physics* 443 (Oct. 15, 2021), p. 110495. URL: https://www.sciencedirect.com/science/article/pii/S0021999121003909 (visited on 09/25/2023) (cit. on p. 142).

[41] L. Einkemmer, C. Lubich. 'A Low-Rank Projector-Splitting Integrator for the Vlasov–Poisson Equation'. In: *SIAM Journal on Scientific Computing* 40.5 (Jan. 2018), B1330–B1360. URL: https://epubs.siam.org/doi/abs/10.1137/18M116383X (visited on 09/13/2023) (cit. on p. 142).

[42] *Energy Aware Runtime*. Leibniz-Rechenzentrum (LRZ). URL: https://doku.lrz.de/energy-aware-runtime-10746191.html#EnergyAwareRuntime-Changingcpuaffinitywithinyourapplication (visited on 07/07/2023) (cit. on p. 93).

[43] EUROfusion. *European researchers achieve fusion energy record*. URL: https://www.euro-fusion.org/news/2022/european-researchers-achieve-fusion-energy-record/ (visited on 04/01/2022) (cit. on p. 11).

[44] M. Fernando, D. Neilsen, E. Hirschmann, Y. Zlochower, H. Sundar, O. Ghattas, G. Biros. 'A GPU-Accelerated AMR Solver for Gravitational Wave Propagation'. In: *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. SC22: International Conference for High Performance Computing, Networking, Storage and Analysis. Nov. 2022, pp. 1–15 (cit. on p. 67).

[45] M. Fernando, D. Neilsen, E. W. Hirschmann, H. Sundar. 'A Scalable Framework for Adaptive Computational General Relativity on Heterogeneous Clusters'. In: *Proceedings of the ACM International Conference on Supercomputing*. ICS '19. New

York, NY, USA: Association for Computing Machinery, June 26, 2019, pp. 1–12. URL: https://dl.acm.org/doi/10.1145/3330345.3330346 (visited on 08/01/2023) (cit. on p. 67).

[46] M. Fernando, D. Neilsen, Y. Zlochower, E. W. Hirschmann, H. Sundar. 'Massively Parallel Simulations of Binary Black Holes with Adaptive Wavelet Multiresolution'. In: *Physical Review D* 107.6 (Mar. 17, 2023), p. 064035. URL: https://link.aps.org/doi/10.1103/PhysRevD.107.064035 (visited on 07/05/2023) (cit. on p. 67).

[47] C. Feuersänger. *Pgfplots*. URL: https://ctan.org/pkg/pgfplots (visited on 08/31/2023) (cit. on p. 146).

[48] I. Foster, Y. Zhao, I. Raicu, S. Lu. 'Cloud Computing and Grid Computing 360-Degree Compared'. In: *2008 Grid Computing Environments Workshop*. 2008 Grid Computing Environments Workshop. ISSN: 2152-1093. Nov. 2008, pp. 1–10 (cit. on p. 67).

[49] F. S. Foundation. *GNU Lesser General Public License v3.0*. GNU Project. URL: https://www.gnu.org/licenses/lgpl-3.0.html (visited on 08/01/2023) (cit. on p. 68).

[50] *Fritz Parallel Cluster (NHR+Tier3)*. URL: https://hpc.fau.de/systems-services/documentation-instructions/clusters/fritz-cluster/ (visited on 06/30/2023) (cit. on pp. 92, 93).

[51] J. Garcke. 'Sparse Grids in a Nutshell'. In: *Sparse Grids and Applications*. Ed. by J. Garcke, M. Griebel. Lecture Notes in Computational Science and Engineering. Berlin, Heidelberg: Springer, 2013, pp. 57–80 (cit. on pp. 29, 30).

[52] K. Germaschewski, B. Allen, T. Dannert, M. Hrywniak, J. Donaghy, G. Merlo, S. Ethier, E. D'Azevedo, F. Jenko, A. Bhattacharjee. 'Toward exascale whole-device modeling of fusion devices: Porting the GENE gyrokinetic microturbulence code to GPU'. In: *Physics of Plasmas* 28.6 (June 1, 2021). Publisher: American Institute of Physics, p. 062501. URL: https://aip.scitation.org/doi/abs/10.1063/5.0046327 (visited on 12/14/2021) (cit. on pp. 98, 102).

[53] T. Gerstner, M. Griebel. 'Dimension–adaptive tensor–product quadrature'. In: *Computing* 71.1 (2003), pp. 65–87 (cit. on pp. 29, 35, 67).

[54] A. Ghizzo, D. Del Sarto, M. Sarrat. 'Low- and high-frequency nature of oblique filamentation modes. I. Linear theory'. In: *Physics of Plasmas* 27.7 (July 21, 2020), p. 072103. URL: https://doi.org/10.1063/5.0003697 (visited on 04/26/2023) (cit. on p. 14).

[55] *GitHub Copilot · Your AI Pair Programmer*. GitHub. URL: https://github.com/features/copilot/ (visited on 08/31/2023) (cit. on p. 146).

[56] T. Görler, X. Lapillonne, S. Brunner, T. Dannert, F. Jenko, F. Merz, D. Told. 'The Global Version of the Gyrokinetic Turbulence Code GENE'. In: *Journal of Computational Physics* 230.18 (Aug. 1, 2011), pp. 7053–7071. URL: `https://www.sciencedirect.com/science/article/pii/S0021999111003457` (visited on 08/31/2023) (cit. on p. 91).

[57] V. Grandgirard, J. Abiteboul, J. Bigot, T. Cartier-Michaud, N. Crouseilles, G. Dif-Pradalier, C. Ehrlacher, D. Esteve, X. Garbet, P. Ghendrih. 'A 5D Gyrokinetic Full-f Global Semi-Lagrangian Code for Flux-Driven Ion Turbulence Simulations'. In: *Computer physics communications* 207 (2016), pp. 35–68 (cit. on p. 142).

[58] V. Grandgirard, Y. Sarazin, X. Garbet, G. Dif-Pradalier, P. Ghendrih, N. Crouseilles, G. Latu, E. Sonnendrücker, N. Besse, P. Bertrand. 'GYSELA, a Full-f Global Gyrokinetic Semi-Lagrangian Code for ITG Turbulence Simulations'. In: *Aip Conference Proceedings*. Vol. 871. 1. American Institute of Physics, 2006, pp. 100–111 (cit. on p. 142).

[59] M. Griebel, W. Huber, U. Rüde, 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, D. Trystram. Vol. 634. LNCS. 1992 (cit. on pp. 31, 80).

[60] M. Griebel, F. Koster. 'Multiscale Methods for the Simulation of Turbulent Flows'. In: *Numerical Flow Simulation III*. Ed. by E. H. Hirschel. Notes on Numerical Fluid Mechanics and Multidisciplinary Design (NNFM). Berlin, Heidelberg: Springer, 2003, pp. 203–214 (cit. on p. 38).

[61] M. Griebel, M. Schneider, C. Zenger. 'A combination technique for the solution of sparse grid problems'. In: *Proceedings of the IMACS International Symposium on Iterative Methods in Linear Algebra: Brussels, Belgium, 2 - 4 April, 1991*. Ed. by P. de Groen, R. Beauwens. Auch als SFB Bericht 342/19/90 A, Institut für Informatik, TU München, 1990. IMACS. North Holland, 1992 (cit. on pp. 15, 29, 31).

[62] M. Griebel, M. A. Schweitzer, L. Troska. 'A Dimension-Oblivious Domain Decomposition Method Based on Space-Filling Curves'. In: *SIAM Journal on Scientific Computing* 45.2 (Apr. 30, 2023), A369–A396. URL: `https://epubs.siam.org/doi/full/10.1137/21M1454481` (visited on 09/26/2023) (cit. on p. 142).

[63] M. Griebel, V. Thurner. 'The Efficient Solution of Fluid Dynamics Problems by the Combination Technique'. In: *International Journal of Numerical Methods for Heat & Fluid Flow* 5.3 (Jan. 1, 1995), pp. 251–269. URL: `https://doi.org/10.1108/EUM0000000004119` (visited on 04/04/2023) (cit. on p. 21).

[64] W. Guo, Y. Cheng. 'A Sparse Grid Discontinuous Galerkin Method for High-Dimensional Transport Equations and Its Application to Kinetic Simulations'. In: *SIAM Journal on Scientific Computing* 38.6 (Jan. 1, 2016), A3381–A3409. URL: `https://epubs.siam.org/doi/10.1137/16M1060017` (visited on 08/08/2019) (cit. on pp. 38–40).

[65] M. Gutnic, M. Haefele, E. Sonnendrücker. 'Moments conservation in adaptive Vlasov solver'. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*. Proceedings of the 8th International Computational Accelerator Physics Conference 558.1 (Mar. 1, 2006), pp. 159–162. URL: `https://www.sciencedirect.com/science/article/pii/S0168900205021327` (visited on 06/15/2023) (cit. on p. 39).

[66] M. Gutnic, M. Haefele, I. Paun, E. Sonnendrücker. 'Vlasov simulations on an adaptive phase-space grid'. In: *Computer Physics Communications*. Proceedings of the 18th International Conferene on the Numerical Simulation of Plasmas 164.1 (Dec. 1, 2004), pp. 214–219. URL: `https://www.sciencedirect.com/science/article/pii/S0010465504002838` (visited on 07/18/2022) (cit. on p. 39).

[67] W. Hackbusch. *Multi-Grid Methods and Applications*. Vol. 4. Springer Series in Computational Mathematics. Berlin, Heidelberg: Springer, 1985. URL: `http://link.springer.com/10.1007/978-3-662-02427-0` (visited on 07/27/2023) (cit. on pp. 25, 26).

[68] M. Haefele, G. Latu, M. Gutnic. 'A parallel Vlasov solver using a wavelet based adaptive mesh refinement'. In: *2005 International Conference on Parallel Processing Workshops (ICPPW'05)*. 2005 International Conference on Parallel Processing Workshops (ICPPW'05). ISSN: 2332-5690. June 2005, pp. 181–188 (cit. on p. 39).

[69] G. Hager. *Georg Hager's Blog | Fooling the Masses – Stunt 5: Instead of Performance, Plot Absolute Runtime versus CPU Count!* Sept. 24, 2010. URL: `http://blogs.fau.de/hager/archives/6130` (visited on 08/10/2023) (cit. on p. 116).

[70] G. Hager, G. Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. Boca Raton: CRC Press, Aug. 17, 2010. 356 pp. (cit. on pp. 62, 66, 111).

[71] B. Harding. 'Adaptive Sparse Grids and Extrapolation Techniques'. In: *Sparse Grids and Applications - Stuttgart 2014*. Ed. by J. Garcke, D. Pflüger. Lecture Notes in Computational Science and Engineering. Cham: Springer International Publishing, 2016, pp. 79–102 (cit. on p. 30).

[72] *Hardware of SuperMUC-NG*. Leibniz-Rechenzentrum (LRZ). URL: `https://doku.lrz.de/display/PUBLIC/Hardware+of+SuperMUC-NG` (visited on 04/05/2023) (cit. on pp. 92, 93).

[73] *Hawk Hardware and Architecture - HLRS Platforms*. URL: https://kb.hlrs.de/platforms/index.php/Hawk_Hardware_and_Architecture (visited on 07/05/2023) (cit. on p. 93).

[74] 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 (cit. on pp. 62, 66, 68, 72, 92, 99, 111).

[75] M. Heene, A. P. Hinojosa, M. Obersteiner, H.-J. Bungartz, 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*. Springer, 2018, pp. 513–529 (cit. on pp. 66, 68).

[76] M. Heene, C. Kowitz, D. Pflüger. 'Load Balancing for Massively Parallel Computations with the Sparse Grid Combination Technique.' In: *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*. Ed. by M. Bader, A. Bode, H.-J. Bungartz, M. Gerndt, G. R. Joubert, F. Peters. Vol. 25. Advances in Parallel Computing. 2014, pp. 574–583 (cit. on pp. 69, 71, 126).

[77] M. Heene, D. Pflüger. 'Scalable Algorithms for the Solution of Higher-Dimensional PDEs'. In: *Software for Exascale Computing - SPPEXA 2013-2015*. Ed. by H.-J. Bungartz, P. Neumann, W. E. Nagel. Lecture Notes in Computational Science and Engineering. Springer International Publishing, 2016, pp. 165–186 (cit. on p. 66).

[78] M. Henon. 'Vlasov Equation'. In: *Astronomy and Astrophysics* 114 (Oct. 1, 1982), p. 211. URL: https://ui.adsabs.harvard.edu/abs/1982A&A...114..211H (visited on 08/08/2023) (cit. on p. 12).

[79] S. Hirschmann. 'Load-Balancing for Scalable Simulations with Large Particle Numbers'. doctoralThesis. 2021. URL: http://elib.uni-stuttgart.de/handle/11682/11813 (visited on 07/28/2023) (cit. on p. 96).

[80] *HLRS High Performance Computing Center Stuttgart: HPE Apollo (Hawk)*. URL: https://www.hlrs.de/solutions/systems/hpe-apollo-hawk (visited on 07/03/2023) (cit. on pp. 92, 93).

[81] *HPC in Germany - Who? What? Where? - HPC in Germany - Gauß-Allianz*. archived in https://web.archive.org/web/20230228030921/https://gauss-allianz.de/en/hpc-ecosystem. URL: https://gauss-allianz.de/en/hpc-ecosystem (visited on 08/07/2023) (cit. on pp. 14, 92).

[82] *HPE Hawk Hardware and Architecture - HLRS Platforms*. URL: https://kb.hlrs.de/platforms/index.php/HPE_Hawk_Hardware_and_Architecture (visited on 08/17/2021) (cit. on p. 93).

[83] W. Huber. *Turbulenzsimulation mit der Kombinationsmethode auf Workstation-Netzen und Parallelrechnern*. Herbert Utz Verlag, 1996. 164 pp. URL: `https://www5.in.tum.de/publikat/diss/huberw.ps.gz` (cit. on p. 38).

[84] P. Hupp, R. Jacob. 'A Cache-Optimal Alternative to the Unidirectional Hierarchization Algorithm'. In: *Proceedings of the Workshop Sparse Grids and Algorithms 2014* (2015) (cit. on p. 111).

[85] P. Hupp, M. Heene, R. Jacob, D. Pflüger. 'Global Communication Schemes for the Numerical Solution of High-Dimensional PDEs'. In: *Parallel Computing* 52 (Feb. 1, 2016), pp. 78–105. URL: `https://www.sciencedirect.com/science/article/pii/S0167819115001623` (visited on 06/19/2023) (cit. on pp. 72–75).

[86] P. Hupp, M. Heene, R. Jacob, D. Pflüger. 'Global Communication Schemes for the Numerical Solution of High-dimensional PDEs'. In: *Parallel Computing* 52.C (Feb. 2016), pp. 78–105. URL: `http://dx.doi.org/10.1016/j.parco.2015.12.006` (cit. on p. 76).

[87] P. Hupp, R. Jacob, M. Heene, D. Pflüger, M. Hegland. 'Global Communication Schemes for the Sparse Grid Combination Technique'. In: *Parallel Computing: Accelerating Computational Science and Engineering (CSE)* (2014), pp. 564–573. URL: `https://ebooks.iospress.nl/doi/10.3233/978-1-61499-381-0-564` (visited on 03/14/2023) (cit. on pp. 72, 74).

[88] *Hyteg / Hyteg · GitLab*. Aug. 1, 2023. URL: `https://i10git.cs.fau.de/hyteg/hyteg` (visited on 08/07/2023) (cit. on p. 67).

[89] F. Jenko, D. Told, T. Görler, J. Citrin, A. B. Navarro, C. Bourdelle, S. Brunner, G. Conway, T. Dannert, H. Doerk, D. R. Hatch, J. W. Haverkort, J. Hobirk, G. M. D. Hogeweij, P. Mantica, M. J. Pueschel, O. Sauter, L. Villard, E. Wolfrum. 'Global and local gyrokinetic simulations of high-performance discharges in view of ITER'. In: *Nuclear Fusion* 53.7 (May 2013), p. 073003. URL: `https://doi.org/10.1088%2F0029-5515%2F53%2F7%2F073003` (visited on 03/19/2019) (cit. on pp. 91, 98, 99).

[90] G. Karypis, V. Kumar. 'A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs'. In: *SIAM Journal on Scientific Computing* 20.1 (Jan. 1998). Publisher: Society for Industrial and Applied Mathematics, pp. 359–392. URL: `https://epubs.siam.org/doi/abs/10.1137/S1064827595287997` (visited on 03/14/2023) (cit. on p. 86).

[91] N. K.-R. Kevlahan. 'Adaptive Wavelet Methods for Earth Systems Modelling'. In: *Fluids* 6.7 (July 2021). Number: 7 Publisher: Multidisciplinary Digital Publishing Institute, p. 236. URL: `https://www.mdpi.com/2311-5521/6/7/236` (visited on 06/15/2023) (cit. on p. 39).

[92] S. Knapek. ‘Approximation und Kompression mit Tensorprodukt-Multiskalenräumen’. Dissertation. Universität Bonn, Apr. 2000 (cit. on pp. 29, 51).

[93] T. Koprucki, M. Kohlhase, K. Tabelow, D. Müller, F. Rabe. ‘Model pathway diagrams for the representation of mathematical models’. In: *Optical and Quantum Electronics* 50.2 (Jan. 23, 2018), p. 70. URL: `https://doi.org/10.1007/s11082-018-1321-7` (visited on 08/31/2022) (cit. on pp. 13, 47).

[94] K. Kormann, K. Reuter, M. Rampp. ‘A massively parallel semi-Lagrangian solver for the six-dimensional Vlasov–Poisson equation’. In: *The International Journal of High Performance Computing Applications* (Mar. 27, 2019). URL: `https://doi.org/10.1177/1094342019834644` (cit. on pp. 48, 54, 169).

[95] K. Kormann, E. Sonnendrücker. ‘Sparse Grids for the Vlasov–Poisson Equation’. In: *Sparse Grids and Applications - Stuttgart 2014*. Ed. by J. Garcke, D. Pflüger. Lecture Notes in Computational Science and Engineering. Springer International Publishing, 2016, pp. 163–190 (cit. on pp. 38, 57).

[96] M. Kosiol. *The Sparse Grid Combination Technique for PDE Solutions of Higher Order*. Projektarbeit SimTech. Universität Stuttgart, Oct. 14, 2022, p. 27 (cit. on p. 38).

[97] F. Koster. ‘Multiskalen-basierte Finite-Differenzen-Verfahren auf adaptiven dünnen Gittern’. Thesis. 2002. URL: `https://bonndoc.ulb.uni-bonn.de/xmlui/handle/20.500.11811/1696` (visited on 02/23/2022) (cit. on pp. 24, 25, 27, 37, 38).

[98] P. Kovesi. *Good Colour Maps: How to Design Them*. Sept. 11, 2015. arXiv: `1509.03700 [cs]`. URL: `http://arxiv.org/abs/1509.03700` (visited on 08/09/2023). preprint (cit. on p. 146).

[99] C. Kowitz. ‘Applying the Sparse Grid Combination Technique in Linear Gyrokinetics’. Dissertation. München: Technische Universität München, 2016 (cit. on pp. 48, 100).

[100] C. Kowitz, M. Hegland. ‘The Sparse Grid Combination Technique for Computing Eigenvalues in Linear Gyrokinetics’. In: *Procedia Computer Science* 18.0 (2013). 2013 International Conference on Computational Science, pp. 449–458. URL: `http://www.sciencedirect.com/science/article/pii/S1877050913003517` (cit. on pp. 38, 98).

[101] C. Kowitz, D. Pflüger, F. Jenko, M. Hegland. ‘The Combination Technique for the Initial Value Problem in Linear Gyrokinetics’. In: *Sparse Grids and Applications*. Ed. by J. Garcke, M. Griebel. Lecture Notes in Computational Science and Engineering. Berlin, Heidelberg: Springer, 2013, pp. 205–222 (cit. on p. 38).

[102] 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'. In: *Software for Exascale Computing - SPPEXA 2016-2019*. Lecture Notes in Computational Science and Engineering. Cham: Springer International Publishing, Jan. 1, 2020, pp. 301–329 (cit. on pp. 18, 38, 62).

[103] L. D. Landau. *Oscillations of an Electron Plasma*. Google-Books-ID: D0CSDInzokMC. 1946. 22 pp. (cit. on p. 54).

[104] B. Lastdrager. *Numerical time integration on sparse grids*. Universiteit van Amsterdam [Host], 2002 (cit. on pp. 34, 52).

[105] *LaTeX Template for Scientific Theses*. latextemplates, Aug. 16, 2023. URL: `https://github.com/latextemplates/scientific-thesis-template` (visited on 08/31/2023) (cit. on p. 146).

[106] *Leinkemmer / Sldg — Bitbucket*. URL: `https://bitbucket.org/leinkemmer/sldg/src/master/` (visited on 06/28/2022) (cit. on p. 142).

[107] X. Litaudon, F. Jenko, D. Borba, D. V. Borodin, B. J. Braams, S. Brezinsek, I. Calvo, R. Coelho, A. J. H. Donné, O. Embréus, D. Farina, T. Görler, J. P. Graves, R. Hatzky, J. Hillesheim, F. Imbeaux, D. Kalupin, R. Kamendje, H.-T. Kim, H. Meyer, F. Militello, K. Nordlund, C. Roach, F. Robin, M. Romanelli, F. Schluck, E. Serre, E. Sonnendrücker, P. Strand, P. Tamain, D. Tskhakaya, J. L. Velasco, L. Villard, S. Wiesen, H. Wilson, F. Zonca. 'EUROfusion-theory and advanced simulation coordination (E-TASC): programme and the role of high performance computing'. In: *Plasma Physics and Controlled Fusion* 64.3 (Feb. 2022), p. 034005. URL: `https://dx.doi.org/10.1088/1361-6587/ac44e4` (visited on 05/08/2023) (cit. on pp. 65, 139).

[108] G. Manfredi. 'Long-Time Behavior of Nonlinear Landau Damping'. In: *Physical Review Letters* 79.15 (Oct. 13, 1997). Publisher: American Physical Society, pp. 2815–2818. URL: `https://link.aps.org/doi/10.1103/PhysRevLett.79.2815` (visited on 01/17/2022) (cit. on p. 54).

[109] R. Margraf. *A Brief History of U.S. Funding of Fusion Energy*. Mar. 27, 2021. URL: `http://large.stanford.edu/courses/2021/ph241/margraf1/` (visited on 04/26/2023) (cit. on p. 11).

[110] S. Matsuoka, J. Domke, M. Wahib, A. Drozd, T. Hoefler. 'Myths and Legends in High-Performance Computing'. In: *The International Journal of High Performance Computing Applications* (Apr. 24, 2023), p. 10943420231166608. URL: `https://doi.org/10.1177/10943420231166608` (visited on 06/19/2023) (cit. on pp. 65, 66).

[111] B. May. *How It Works — Sshuttle 1.1.1 Documentation*. archived in https://web.archive.org/web/20230323092842/https://sshuttle.readthedocs.io/en/stable/how-it-works.html. URL: https://sshuttle.readthedocs.io/en/stable/how-it-works.html (visited on 07/27/2023) (cit. on p. 82).

[112] M. Molzer. 'Implementation of a Parallel Sparse Grid Combination Technique for Variable Process Group Sizes'. Bachelor's thesis. TU München, Jan. 2018 (cit. on p. 105).

[113] MPI Forum. *MPI: A Message-Passing Interface Standard. Version 3.1*. 2015. URL: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf (cit. on pp. 68, 71, 83, 88).

[114] P. Munch, K. Kormann, M. Kronbichler. 'hyper.deal: An efficient, matrix-free finite-element library for high-dimensional partial differential equations'. In: *arXiv:2002.08110 [cs, math]* (Feb. 19, 2020). arXiv: 2002.08110. URL: http://arxiv.org/abs/2002.08110 (visited on 08/19/2021) (cit. on pp. 88, 142).

[115] S. Muralikrishnan, A. J. Cerfon, M. Frey, L. F. Ricketson, A. Adelmann. 'Sparse grid-based adaptive noise reduction strategy for particle-in-cell schemes'. In: *Journal of Computational Physics: X* 11 (June 1, 2021), p. 100094. URL: https://www.sciencedirect.com/science/article/pii/S2590055221000111 (visited on 03/18/2022) (cit. on pp. 12, 67).

[116] A. Narayan, J. D. Jakeman. 'Adaptive Leja Sparse Grid Constructions for Stochastic Collocation and High-Dimensional Approximation'. In: *SIAM Journal on Scientific Computing* 36.6 (Jan. 2014), A2952–A2983. URL: https://epubs.siam.org/doi/abs/10.1137/140966368 (visited on 08/09/2023) (cit. on p. 24).

[117] W. M. Nevins, G. W. Hammett, A. M. Dimits, W. Dorland, D. E. Shumaker. 'Discrete particle noise in particle-in-cell simulations of plasma microturbulence'. In: *Physics of Plasmas* 12.12 (Dec. 2005). Publisher: American Institute of Physics, p. 122305. URL: https://aip.scitation.org/doi/full/10.1063/1.2118729 (visited on 04/06/2023) (cit. on p. 12).

[118] E. Novak, K. Ritter. 'High Dimensional Integration of Smooth Functions over Cubes'. In: *Numerische Mathematik* 75.1 (Nov. 1, 1996), pp. 79–97. URL: https://doi.org/10.1007/s002110050231 (visited on 08/09/2023) (cit. on p. 24).

[119] M. Obersteiner. 'A spatially adaptive and massively parallel implementation of the fault-tolerant combination technique'. Dissertation. Technische Universität München, 2021. URL: https://mediatum.ub.tum.de/doc/1613369/1613369.pdf (visited on 04/01/2022) (cit. on pp. 31, 32, 34, 40, 78, 79).

[120] M. Obersteiner. *sparseSpACE - The Sparse Grid Spatially Adaptive Combination Environment*. May 9, 2023. URL: https://github.com/obersteiner/sparseSpACE (visited on 08/07/2023) (cit. on p. 67).

[121] M. Obersteiner, H.-J. Bungartz. 'A Generalized Spatially Adaptive Sparse Grid Combination Technique with Dimension-wise Refinement'. In: *SIAM Journal on Scientific Computing* 43.4 (Jan. 2021). Publisher: Society for Industrial and Applied Mathematics, A2381–A2403. URL: https://epubs.siam.org/doi/abs/10.1137/20M1325885 (visited on 04/13/2022) (cit. on pp. 35, 67).

[122] M. Obersteiner, A. P. Hinojosa, M. Heene, H.-J. Bungartz, 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*. ScalA '17. New York, NY, USA: ACM, 2017, 2:1–2:8. URL: http://doi.acm.org/10.1145/3148226.3148229 (visited on 08/22/2018) (cit. on p. 66).

[123] OpenMP Architecture Review Board. *OpenMP Standard 4.5*. 2015 (cit. on p. 71).

[124] D. Pflüger. *SG++*. archived in https://web.archive.org/web/20230601152122/https://sgpp.sparsegrids.org/. URL: https://sgpp.sparsegrids.org/ (visited on 08/28/2023) (cit. on pp. 66, 68).

[125] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. Verlag Dr. Hut, 2010 (cit. on pp. 29, 39, 48).

[126] C. Piazzola, L. Tamellini. *Sparse Grids Matlab Kit*. URL: https://sites.google.com/view/sparse-grids-kit (visited on 08/01/2023) (cit. on p. 67).

[127] C. Piazzola, L. Tamellini. *The Sparse Grids Matlab Kit – a Matlab Implementation of Sparse Grids for High-Dimensional Function Approximation and Uncertainty Quantification*. Mar. 17, 2022. arXiv: 2203.09314 [cs, math]. URL: http://arxiv.org/abs/2203.09314 (visited on 03/15/2023). preprint (cit. on p. 67).

[128] T. Pollinger. *DisCoTec-combischeme-utilities*. SG++ development team, Oct. 18, 2022. URL: https://github.com/SGpp/DisCoTec-combischeme-utilities (visited on 03/21/2023) (cit. on pp. 85, 88, 143).

[129] T. Pollinger. *Freifrauvonbleifrei/Discotec-Scaling-Scripts*. Mar. 25, 2022. URL: https://github.com/freifrauvonbleifrei/discotec-scaling-scripts (visited on 08/30/2023) (cit. on p. 143).

[130] T. Pollinger. *Replication Data for: A mass-conserving sparse grid combination technique with biorthogonal hierarchical basis functions for kinetic simulations*. Type: dataset. URL: https://darus.uni-stuttgart.de/dataset.xhtml?persistentId=doi:10.18419/darus-2790 (visited on 05/16/2022) (cit. on p. 143).

[131] T. Pollinger. *Replication Data for: Stable and Mass-Conserving High-Dimensional Simulations with the Sparse Grid Combination Technique for Full HPC Systems and Beyond*. unpublished data set, to be published with the thesis, access via https://darus.uni-stuttgart.de/privateurl.xhtml?token=38ee4d43-534b-45b1-a886-046c18144d02. DaRUS. URL: https://darus.uni-stuttgart.de/dataset.xhtml?persistentId=doi:10.18419/darus-3580 (visited on 08/31/2023) (cit. on p. 143).

[132] T. Pollinger, M. Hurler, M. Obersteiner, D. Pflüger. 'Distributing Higher-Dimensional Simulations Across Compute Systems: A Widely Distributed Combination Technique'. In: *2021 IEEE/ACM International Workshop on Hierarchical Parallelism for Exascale Computing (HiPar)*. 2021 IEEE/ACM International Workshop on Hierarchical Parallelism for Exascale Computing (HiPar). Nov. 2021, pp. 1–9 (cit. on pp. 17, 80, 81, 88).

[133] T. Pollinger, K. Kormann. *Replication Data for: "Scaling the Plasma Simulation While Conserving the Mass" - Poster at PASC '22*. DaRUS, June 28, 2022. URL: https://darus.uni-stuttgart.de/dataset.xhtml?persistentId=doi:10.18419/darus-2784 (visited on 08/31/2023) (cit. on p. 143).

[134] T. Pollinger, K. Kormann, D. Pflüger. *Scaling the Plasma Simulation while Conserving the Mass: A Massively-Parallel Semi-Lagrangian Solver with the Sparse Grid Combination Technique*. Was awarded the Best Poster Award at PASC'22. PASC22, June 25, 2022. URL: https://pasc22.pasc-conference.org/program/schedule/presentation/?id=pos109&sess=sess181 (visited on 07/19/2022) (cit. on pp. 17, 167).

[135] T. Pollinger, D. Pflüger. 'Learning-Based Load Balancing for Massively Parallel Simulations of Hot Fusion Plasmas'. In: *Advances in Parallel Computing* 36 (Parallel Computing: Technology Trends 2020), pp. 137–146. URL: http://doi.org/10.3233/APC200034 (cit. on pp. 18, 69, 71, 99, 100).

[136] T. Pollinger, J. Rentrop, D. Pflüger, K. Kormann. *A mass-conserving sparse grid combination technique with biorthogonal hierarchical basis functions for kinetic simulations*. Sept. 23, 2022. arXiv: 2209.14064[physics]. URL: http://arxiv.org/abs/2209.14064 (visited on 11/11/2022) (cit. on pp. 17, 87).

[137] T. Pollinger, J. Rentrop, D. Pflüger, K. Kormann. 'A Stable and Mass-Conserving Sparse Grid Combination Technique with Biorthogonal Hierarchical Basis Functions for Kinetic Simulations'. In: *Journal of Computational Physics* (July 7, 2023), p. 112338. URL: https://www.sciencedirect.com/science/article/pii/S0021999123004333 (visited on 07/17/2023) (cit. on pp. 17, 19, 22, 26, 27, 29, 38, 49–52, 55, 58, 60, 64, 87, 107).

[138] T. Pollinger, A. Van Craen, C. Niethammer, M. Breyer, D. Pflüger. ‘Leveraging the Compute Power of Two HPC Systems for Higher-Dimensional Grid-Based Simulations with the Widely-Distributed Sparse Grid Combination Technique’. In: SC ’23. Association for Computing Machinery, Nov. 11, 2023. URL: https://dl.acm.org/doi/10.1145/3581784.3607036 (visited on 11/15/2023) (cit. on pp. 17, 78, 80, 81, 83–85, 88, 131, 133, 135).

[139] R. F. Post. ‘Controlled Fusion Research—An Application of the Physics of High Temperature Plasmas’. In: *Reviews of Modern Physics* 28.3 (July 1, 1956). Publisher: American Physical Society, pp. 338–362. URL: https://link.aps.org/doi/10.1103/RevModPhys.28.338 (visited on 04/26/2023) (cit. on p. 11).

[140] *Processor Affinity — JUWELS User Documentation Documentation*. archived in https://web.archive.org/web/20230811070412/https://apps.fz-juelich.de/jsc/hps/juwels/affinity.html. URL: https://apps.fz-juelich.de/jsc/hps/juwels/affinity.html#processor-affinity (visited on 08/11/2023) (cit. on p. 111).

[141] J. D. Regele, O. V. Vasilyev. ‘An adaptive wavelet-collocation method for shock computations’. In: *International Journal of Computational Fluid Dynamics* 23.7 (Aug. 1, 2009), pp. 503–518. URL: https://doi.org/10.1080/10618560903117105 (visited on 06/15/2023) (cit. on p. 39).

[142] M. F. Rehme. ‘B-Splines on Sparse Grids for Uncertainty Quantification’. PhD thesis. Universität Stuttgart, 2021. URL: http://elib.uni-stuttgart.de/handle/11682/11771 (visited on 09/30/2022) (cit. on p. 66).

[143] M. F. Rehme, F. Franzelin, D. Pflüger. ‘B-splines on sparse grids for surrogates in uncertainty quantification’. In: *Reliability Engineering & System Safety* 209 (May 1, 2021), p. 107430. URL: https://www.sciencedirect.com/science/article/pii/S0951832021000016 (visited on 05/08/2023) (cit. on p. 25).

[144] C. Reisinger. ‘Analysis of linear difference schemes in the sparse grid combination technique’. In: *IMA Journal of Numerical Analysis* 33.2 (Apr. 1, 2013), pp. 544–581. URL: https://academic.oup.com/imajna/article/33/2/544/653322 (visited on 08/26/2019) (cit. on p. 34).

[145] L. F. Ricketson, A. J. Cerfon. ‘Sparse grid techniques for particle-in-cell schemes’. In: *Plasma Physics and Controlled Fusion* 59.2 (Dec. 2016). Publisher: IOP Publishing, p. 024002. URL: https://doi.org/10.1088/1361-6587/59/2/024002 (visited on 03/18/2022) (cit. on pp. 12, 32, 67).

[146] K. M. Röhner. ‘Learning from Data with Geometry-Aware Sparse Grids’. Technische Universität München, 2020. URL: https://mediatum.ub.tum.de/604993?query=kilian+r%C3%B6hner&show_id=1554038&srcnodeid=604993 (visited on 08/07/2023) (cit. on p. 66).

[147] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, S. Tarantola. *Global Sensitivity Analysis: The Primer*. John Wiley & Sons, 2008 (cit. on p. 32).

[148] N. Scharping. *Why Nuclear Fusion Is Always 30 Years Away*. Discover Magazine. Mar. 23, 2016. URL: https://www.discovermagazine.com/technology/why-nuclear-fusion-is-always-30-years-away (visited on 05/02/2023) (cit. on p. 11).

[149] B. Schuller, T. Pohlmann, K. Benedyczak, J. Rybicki. *UFTP*. Apr. 17, 2023. URL: https://zenodo.org/record/8043002 (visited on 06/16/2023) (cit. on p. 84).

[150] *Science | Scientists and cloud computing push the boundaries of ITER plasma disruption simulations*. ITER. Apr. 4, 2022. URL: http://www.iter.org/newsline/-/3739 (visited on 05/05/2023) (cit. on p. 12).

[151] SG++ development team. *DisCoTec*. SG++ development team, July 30, 2023. URL: https://github.com/SGpp/DisCoTec (visited on 10/11/2023) (cit. on p. 68).

[152] S. A. Smolyak. ‘Quadrature and interpolation formulas for tensor products of certain class of functions’. In: *Soviet Mathematics Doklady* 4 (1963). Russian Original: Doklady Akademii Nauk SSSR, 148 (5): 1042–1053, pp. 240–243 (cit. on p. 29).

[153] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer. *June 2023 | TOP500*. URL: https://www.top500.org/lists/top500/2023/06/ (visited on 07/03/2023) (cit. on pp. 65, 66, 92, 93, 132).

[154] W. Sweldens. ‘The Lifting Scheme: A Construction of Second Generation Wavelets’. In: *SIAM Journal on Mathematical Analysis* 29.2 (Mar. 1, 1998). Publisher: Society for Industrial and Applied Mathematics, pp. 511–546. URL: https://epubs.siam.org/doi/abs/10.1137/S0036141095289051 (visited on 01/14/2022) (cit. on pp. 24, 26, 27, 141).

[155] H. Tajfel, M. G. Billig, R. P. Bundy, C. Flament. ‘Social Categorization and Intergroup Behaviour’. In: *European Journal of Social Psychology* 1.2 (1971), pp. 149–178. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/ejsp.2420010202 (visited on 07/14/2023) (cit. on p. 77).

[156] T. Tantau, et al. *TikZ & PGF*. 2010. URL: https://ctan.org/pkg/pgf (visited on 08/31/2023) (cit. on p. 146).

[157] Z. Tao, W. Guo, Y. Cheng. 'Sparse grid discontinuous Galerkin methods for the Vlasov-Maxwell system'. In: *Journal of Computational Physics: X* 3 (2019), p. 100022 (cit. on pp. 38, 141).

[158] S. development team. *Welcome to SG++*. Aug. 22, 2022. URL: `https://github.com/SGpp/SGpp` (visited on 09/01/2022) (cit. on pp. 66, 68).

[159] *TerraNeo*. TerraNeo: Integrated Co-Design of an Exascale Earth Mantle Modeling Framework. URL: `https://www.terraneo.fau.de/` (visited on 08/07/2023) (cit. on p. 67).

[160] The GENE Development Team. *The GENE Code*. URL: `http://genecode.org/` (visited on 08/27/2019) (cit. on pp. 91, 98).

[161] *Turbulence simulations reveal promising insight for fusion*. ITER. Apr. 10, 2014. URL: `http://www.iter.org/newsline/-/1888` (visited on 05/05/2023) (cit. on p. 12).

[162] J. Valentin. 'B-Splines for Sparse Grids: Algorithms and Application to Higher-Dimensional Optimization'. PhD thesis. University of Stuttgart, 2019 (cit. on pp. 25, 28, 29, 66, 70, 123).

[163] J. Valentin. *Original Repository of Julian Valentin's PhD Thesis*. original-date: 2022-01-08T08:51:16Z. Jan. 15, 2022. URL: `https://github.com/valentjn/thesis` (visited on 03/02/2022) (cit. on p. 28).

[164] K. Vladimir. *Anatoly Vlasov Heritage: 60-Year-Old Controversy*. 2023. URL: `https://philpapers.org/rec/VLAAVH` (visited on 08/08/2023) (cit. on p. 12).

[165] A. A. Vlasov. 'The Vibrational Properties of an Electron Gas'. In: *Soviet Physics Uspekhi* 10.6 (June 30, 1968), p. 721. URL: `https://iopscience.iop.org/article/10.1070/PU1968v010n06ABEH003709/meta` (visited on 08/08/2023) (cit. on pp. 12, 13).

[166] Z. Wang, Q. Tang, W. Guo, Y. Cheng. 'Sparse grid discontinuous Galerkin methods for high-dimensional elliptic equations'. In: *Journal of Computational Physics* 314 (2016), pp. 244–263 (cit. on p. 38).

[167] R. D. White, R. E. Robson, S. Dujko, P. Nicoletopoulos, B. Li. 'Recent Advances in the Application of Boltzmann Equation and Fluid Equation Methods to Charged Particle Transport in Non-Equilibrium Plasmas'. In: *Journal of Physics D: Applied Physics* 42.19 (Sept. 2009), p. 194001. URL: `https://dx.doi.org/10.1088/0022-3727/42/19/194001` (visited on 04/06/2023) (cit. on p. 12).

[168] F. Wilms, A. B. Navarro, G. Merlo, L. Leppin, T. Görler, T. Dannert, F. Hindenlang, F. Jenko. 'Global electromagnetic turbulence simulations of W7-X-like plasmas with GENE-3D'. In: *Journal of Plasma Physics* 87.6 (Dec. 2021). Publisher: Cambridge University Press. URL: https://www.cambridge.org/core/journals/journal-of-plasma-physics/article/global-electromagnetic-turbulence-simulations-of-w7xlike-plasmas-with-gene3d/AFF0F24A1A52D397D7983BAB2E872E9F (visited on 03/29/2022) (cit. on p. 98).

[169] M. Wittmann, G. Hager, G. Wellein. 'Multicore-Aware Parallel Temporal Blocking of Stencil Codes for Shared and Distributed Memory'. In: *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*. 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW). Apr. 2010, pp. 1–7 (cit. on p. 111).

[170] M. Wolff. *Heaptrack - A Heap Memory Profiler for Linux*. Milian Wolff, Dec. 2, 2014. URL: https://milianw.de/blog/heaptrack-a-heap-memory-profiler-for-linux.html (visited on 08/07/2023) (cit. on p. 169).

[171] J. Wright. *Siunitx*. URL: https://ctan.org/pkg/siunitx (visited on 08/31/2023) (cit. on p. 146).

[172] A. Zeiser. 'Sparse Grid Time-Discontinuous Galerkin Method with Streamline Diffusion for Transport Equations'. In: *Partial Differential Equations and Applications* 4.4 (Aug. 1, 2023), p. 38. URL: https://doi.org/10.1007/s42985-023-00250-2 (visited on 08/08/2023) (cit. on pp. 38, 62).

[173] C. Zenger. 'Sparse Grids'. In: *Notes on Numerical Fluid Mechanics* 31 (1991), pp. 241–251 (cit. on p. 27).

[174] A. N. Ziogas, T. Ben-Nun, G. I. Fernández, T. Schneider, M. Luisier, T. Hoefler. 'A Data-Centric Approach to Extreme-Scale Ab Initio Dissipative Quantum Transport Simulations'. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '19. New York, NY, USA: Association for Computing Machinery, Nov. 17, 2019, pp. 1–13. URL: https://dl.acm.org/doi/10.1145/3295500.3357156 (visited on 05/08/2023) (cit. on p. 66).

Appendix A

Appendix

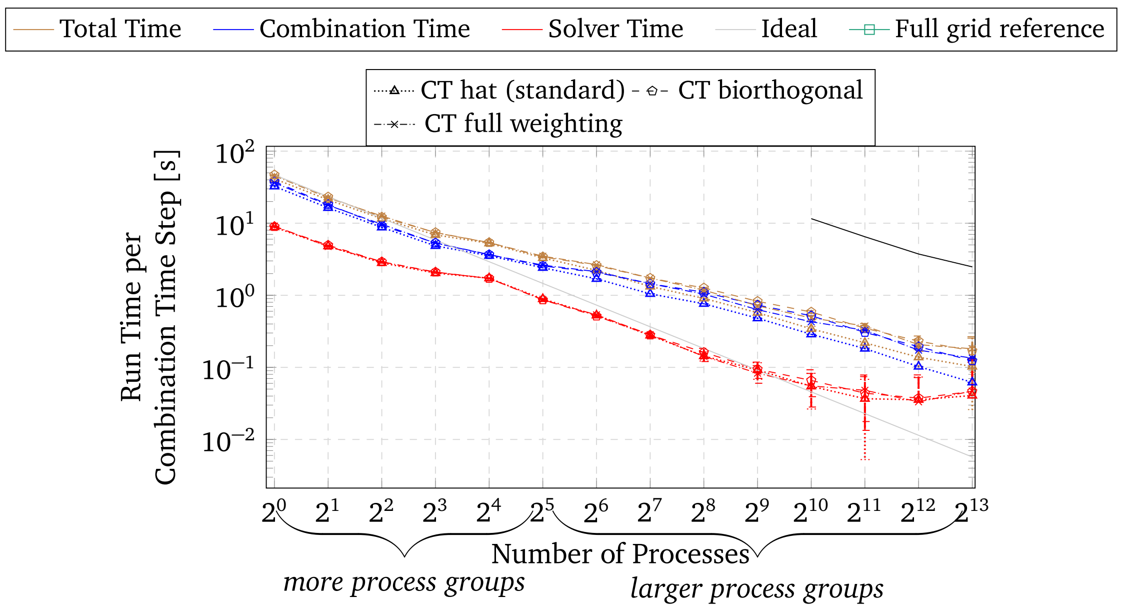
A.1. Comparison: DisCoTec and SeLaLib Run Times 'Old' and Revisited

To evaluate the joint scaling capabilities of DisCoTec and SeLaLib, strong scaling measurements were taken for the two-stream instability problem on HAWK. As the combination scheme considered in Section 3.5.3 is relatively small—only 1.1 GiB are used for the full grid functions in total—the problem was run on up to 8192 ranks (no OpenMP), i. e., 64 nodes. This was considered to be too small to be interesting for Chapter 5. But since it was used as part of a poster [134] and re-run in July 2023, with new DisCoTec features, it can be interesting to see how the run times changed.

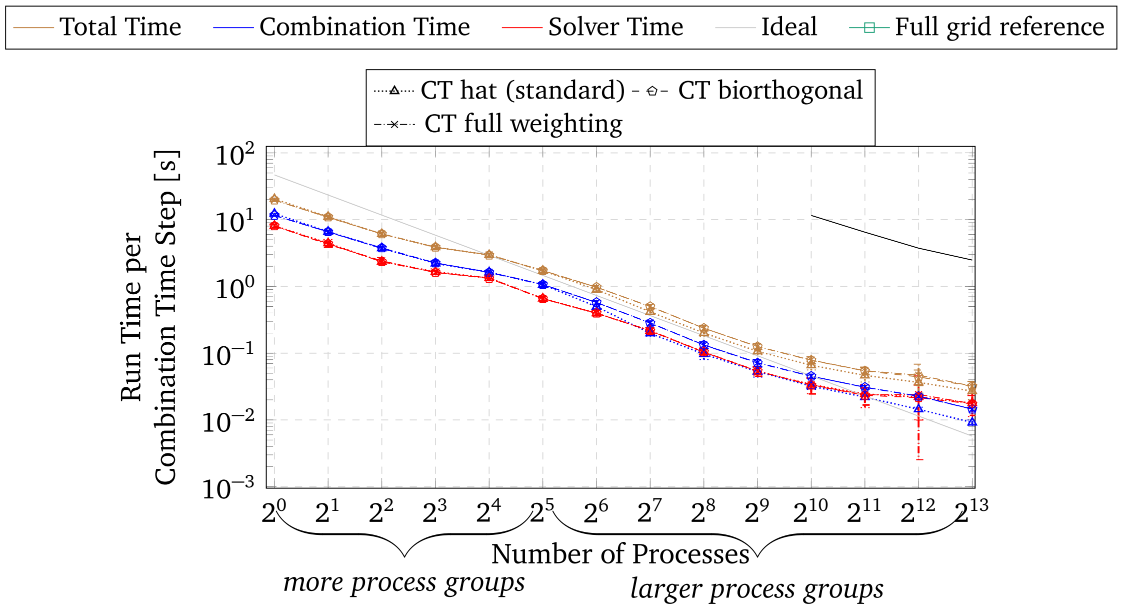
The changes included running without the manager rank (and thus round-robin assignment instead of dynamic load balancing), ChunkedOutgroupSGReduce instead of sparse grid reduce, avoiding copies between DisCoTec and SeLaLib, and using Kahan summation instead of naive summation (which should increase the accuracy of the results but incur some run time overhead). However, it goes beyond the scope of this work to disseminate exactly which optimization played which part in the improved scaling behavior.

The results are shown in Figure A.1.

First, we want to expand upon the discussion in Section 3.6, where the differences in computational cost between the different basis functions were assessed. For both the previous and new results, there is a tiny difference for the combination with the different basis functions, where the time for the mass-conserving functions $\psi^{\texttt{bo}}$ and $\psi^{\texttt{fw}}$ is typically the same. As the number of process groups is increased up to



(a) 'Old' version of DisCoTec with SeLaLib



(b) Revised version of DisCoTec with SeLaLib

Figure A.1.: Comparison of strong scaling timings for DisCoTec with SeLaLib on the two-stream instability problem. Note that the grey 'ideal' scaling line is the exact same between the two plots. The reference solution at a resolution level of $\vec{\ell}^{\max}$ (black line) could only be computed starting at 1024 ranks, as before there was not sufficient main memory available.

32, the times decrease symmetrically for both the standard hat functions and the mass-conserving functions. In this left part of the graph, scaling losses for SeLaLib are due to load imbalances between the process groups. Only when the size of process groups increases, one starts to observe a widening gap for the combination with the different basis transformation operators. This conforms to the expectation, as larger process groups correspond to more spatial domain decomposition, and the communication effects in Section 3.6 become more pronounced.

One can see that the changes in DisCoTec have a significant positive effect on the run times, especially for larger numbers of processes, where the time required per combination was reduced by a factor of approximately 6.8 for the hierarchical hat functions and 9.2 for the mass-conserving basis functions. While the strong scaling efficiency was previously 6% for the combination and 3% for the SeLaLib solver, it is now 16% and 6%, respectively, for the basis transforms with $\psi^{\mathtt{hat}}$.

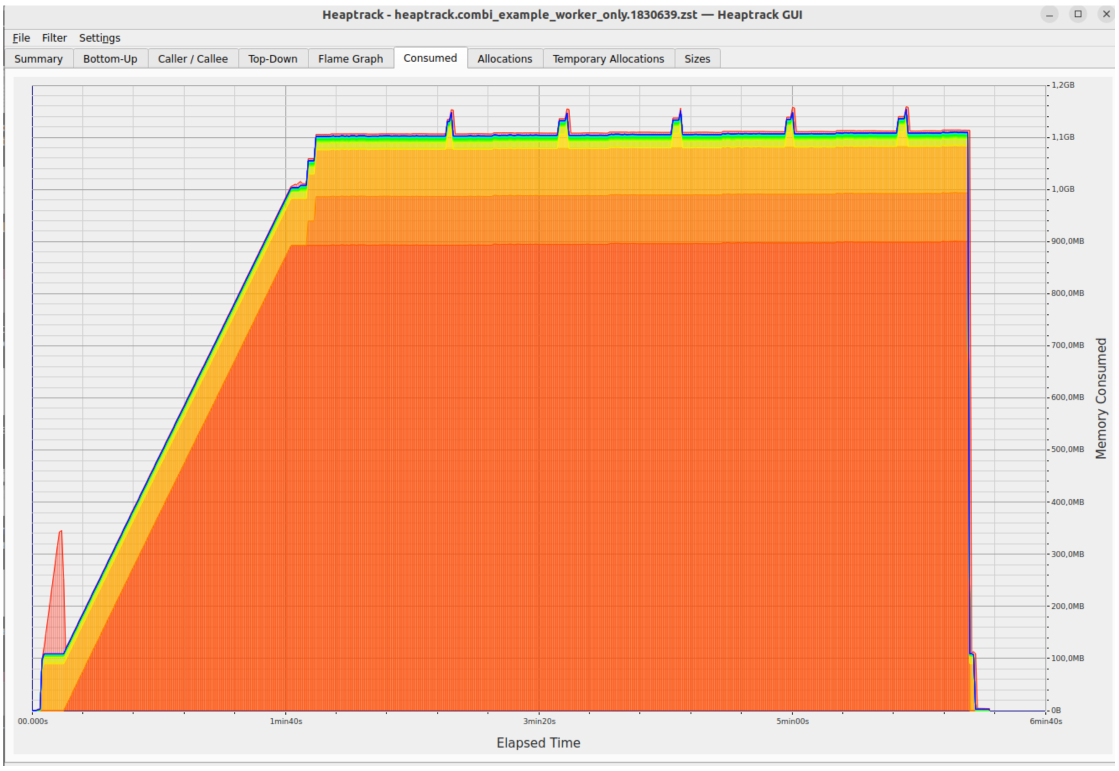
To advance these measurements for larger-scale simulations, it would be particularly interesting to see how using OpenMP shared-memory parallelism may speed up the computation, since SeLaLib supports efficient OpenMP parallelization [94] and DisCoTec now does too, cf. Section 4.7.

A.2. Why Not to Use `MPI_Iallreduce` at Scale

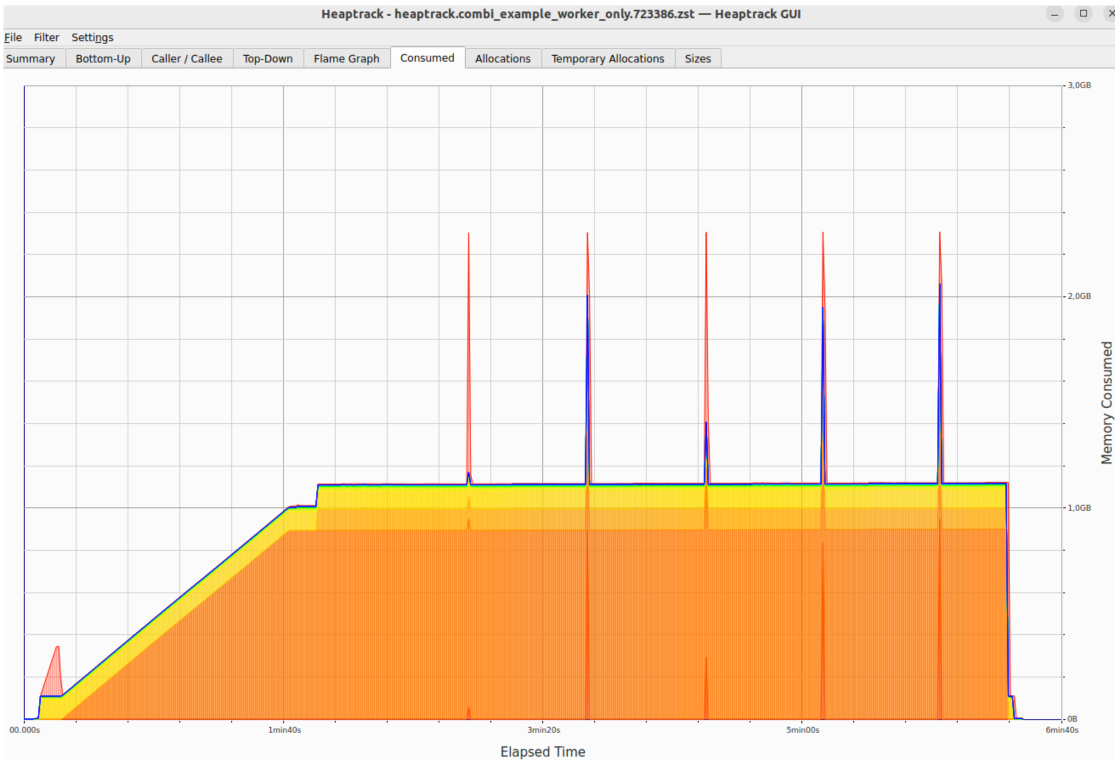
During the development of OutgroupSGReduce in DisCoTec, using non-blocking collectives such as `MPI_Iallreduce` in Algorithm 4.4 was considered. However, this led to unexpected failures of the simulations with both Intel-MPI and OpenMPI. The reason for this were the large memory requirements of the non-blocking collectives.

This could be investigated with the heaptrack memory analyzer tool [170], screenshots of which are shown in Figure A.2. Eight process groups were considered, and ChunkedOutgroupSGReduce was implemented with `MPI_Allreduce` and `MPI_Iallreduce` (followed directly by `MPI_Wait` to obtain the semantics of `MPI_Allreduce`). The size of the sparse grid to be reduced was 190 MiB.

For `MPI_Allreduce`, the memory consumption behaves as expected: The memory consumed prior to the reductions is approximately 1.1 GiB per rank, and one can observe small spikes as the reductions are performed. The magnitude of these spikes is in proportion to the memory chunks granted (16 MiB) for the reduce operation.


Heaptrack - heaptrack.combi_example_worker_only.1830639.zst — Heaptrack GUI
File Filter Settings
Summary Bottom-Up Caller / Callee Top-Down Flame Graph Consumed Allocations Temporary Allocations Sizes
1,2GB
1,1GB
1,0GB
900,0MB
800,0MB
700,0MB
600,0MB
500,0MB
400,0MB
300,0MB
200,0MB
100,0MB
0B
Memory Consumed
00.000s
1min40s
3min20s
5min00s
6min40s
Elapsed Time

(a) . . . with `MPI_Allreduce`


Heaptrack - heaptrack.combi_example_worker_only.723386.zst — Heaptrack GUI
File Filter Settings
Summary Bottom-Up Caller / Callee Top-Down Flame Graph Consumed Allocations Temporary Allocations Sizes
3,0GB
2,0GB
1,0GB
0B
Memory Consumed
00.000s
1min40s
3min20s
5min00s
6min40s
Elapsed Time

(b) . . . with `MPI_Iallreduce` and immediate `MPI_Wait`

Figure A.2.: Heaptrack heap memory consumption screenshots for CT simulations with OutgroupSGReduce, cf. Algorithm 4.4, for a setup with eight process groups of 64 ranks on SuperMUC-NG.

However, for `MPI_Iallreduce`, There are huge intermediate spikes due to temporary allocations (the red lines). Their magnitude is much higher than any of the reduction buffers' size is supposed to be. The fact that they are 'temporary' in heaptrack means that they are freed again immediately, and not used, which makes it unlikely that the allocations happen for performance optimization reasons.

This made the current implementations of the non-blocking reduction unsuitable for use with simulations at severe memory constraints, such as the higher-dimensional simulations considered here.

Glossary and Abbreviations

CT combination technique. 29

Curse of Dimensionality Exponential dependency of the number of DOF on the dimensionality of the problem. 12

dehierarchization Basis transformation from multiscale to nodal basis, also ‘inverse wavelet transform’ in the wavelet context. 21

DOF degree(s) of freedom. 14

hierarchization Basis transformation from nodal to hierarchical/multiscale basis, also ‘wavelet transform’ in the wavelet context. 21

HPC high performance computing. 4, 65

PDE partial differential equation. 4, 12, 15

reduction operation Associative operation on multiple data entries, well suited for parallelization. In the context of this thesis, it usually denotes a summation. For distributed-memory reductions, there is the concept of ‘allreduce’ or `MPI_Allreduce`, if all ranks require the valid result for further computation. 71

SG sparse grid. 27

List of Symbols

Symbol	Name + Description
W	**hierarchical increment space** The discrete function space required to get from a smaller to a larger nodal function space, resulting in a hierarchy of function spaces, cf. Equation (2.3).
$\mathcal{I}$	**index set** Sparse grid description, a plain set of (multi-dimensional) refinement levels for non-spatially-adaptive sparse grids.
$\mathcal{I}^{\mathrm{cj}}$	**(hierarchical) conjoint index set** The set of hierarchical subspaces that two sparse grids have in common..
$\mathcal{I}^{\mathrm{CT}}$	**combination technique index set** The set of nodal function spaces used to construct a sparse grid with the CT, equivalent to the set of component grids, also: combination scheme.
$\mathcal{I}^{\mathrm{SG}}$	**sparse grid index set** The set of hierarchicacal increment spaces contained in a sparse grid.
ℓ	**level** (one-dimensional) refinement level.
$\vec{\ell}$	**level vector** (multi-dimensional) refinement level.
$\vec{\ell}^{\max}$	**maximum level** The highest level vector / the finest resolutions in a combination scheme, in each dimension, respectively.
$\vec{\ell}^{\min}$	**minimum level** The smallest level vector / the coarsest resolutions in a combination scheme, in each dimension, respectively.
ψ	**multiscale basis function** the set of functions that span W, also: wavelet function.
$\psi^{\mathtt{bo}}$	**biorthogonal basis function** A choice of mass-conserving multiscale basis function, defined by $\psi^{\mathtt{bo}} = -\frac{1}{8}\psi^{\mathtt{hat}}(x+1) - \frac{1}{4}\psi^{\mathtt{hat}}(x+\frac{1}{2}) + \frac{3}{4}\psi^{\mathtt{hat}}(x) - \frac{1}{4}\psi^{\mathtt{hat}}(x-\frac{1}{2}) - \frac{1}{8}\psi^{\mathtt{hat}}(x-1)$; also: CDF 3/5 wavelet, dual wavelet to full weighting basis function.

Symbol	Name + Description
$\psi^{\mathtt{fw}}$	**full weighting basis function** A choice of mass-conserving multiscale basis function, no closed form; also: CDF 5/3 wavelet, dual wavelet to biorthogonal basis function.
$\psi^{\mathtt{hat}}$	**hierarchical hat function** $\psi^{\mathtt{hat}}(x) = \max(1-2\left\|x-\frac{1}{2}\right\|, 0)$; also: interpolet of first order, lazy wavelet.
V	**nodal function space** Discrete function space, often interpolating a function at varying refinement levels. In the context of this thesis, V is part of a nested sequence of function spaces containing piecewise linear functions.
ϕ	**scaling function** one of the functions that span V, also: nodal basis function.
$\phi^{\mathtt{hat}}$	**'hat' scaling function** also: linear basis function, nodal hat function.