Load Balancing for Massively Parallel Computations with the Sparse Grid Combination Technique

Mario HEENE\textsuperscript{a,1}, Christoph KOWITZ\textsuperscript{b} and Dirk PFL"UGER\textsuperscript{a}

\textsuperscript{a}Institute for Parallel and Distributed Systems, Universität Stuttgart, Germany
\textsuperscript{b}Institute for Advanced Study, Technische Universität München, Germany

Abstract. Massively parallel simulations of plasma microturbulence using GENE are facing the curse of dimensionality, since the discretization of the five-dimensional gyrokinetic equations requires a large amount of grid points even for only moderate resolutions. The sparse grid combination technique can be used to tackle the curse of dimensionality. Being based on a superposition of anisotropic full grid solutions that can be computed independently of each other, it introduces a second layer of parallelism that will equip GENE for exascale computing. Since the anisotropy of the discretizations of the partial solutions results in massive load imbalances, effective scheduling is crucial in order to exploit this parallelism. In this paper a load model for linear initial value runs with GENE is introduced for effective load balancing for the combination technique.

Keywords. sparse grids, combination technique, load balancing, GENE, exascale computing

1. Introduction

The exponential growth of the computational effort with the number of dimensions, also referred to as the curse of dimensionality, is still a challenge for the solution of higher-dimensional problems. This can be observed in problems like the simulation of the fusion reaction in a tokamak with the gyrokinetic plasma turbulence code GENE, which is based on a five-dimensional (plus time) description of the underlying gyrokinetic equations. Therefore it has so far only been possible to simulate tokamaks of medium size [1].

\textsuperscript{1}Corresponding Author: Mario Heene, Institute for Parallel and Distributed Systems, Universität Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany; Email: mario.heene@ipvs.uni-stuttgart.de
Simulations of the upcoming large fusion experiment ITER will, however, require the performance of exascale systems.

Sparse grids [2] are a hierarchical approach to overcome the curse of dimensionality by drastically reducing the number of unknowns, while preserving a similar accuracy as in the full grid case. Although excellent scaling behavior has been observed applying sparse grids on hybrid multi-core GPU systems [3], due to their recursive and hierarchical structure, the direct sparse grid approach is not applicable for exascale parallelization.

The sparse grid combination technique [4,5] is based on the idea of extrapolating a sparse grid solution by a weighted superposition of coarse and anisotropic full grid solutions. For interpolation the exact sparse grid solution is obtained. For other problems the combination solution might diverge from the direct sparse grid solution. This can be handled by more advanced methods like the optimized combination technique [6]. Instead of solving a single highly coupled problem of size $\mathcal{O}(2^n d^{d-1})$ in the sparse grid space, with the combination technique one can deal with $\mathcal{O}(d n^{d-1})$ independent partial problems of size $\mathcal{O}(2^n)$. Although the overall computational effort is increased, the asynchronous and decoupled computation of the simpler partial solutions minimizes the communication and synchronization overhead, thus making the sparse grid combination technique a promising method for the solution of large-scale higher-dimensional problems on exascale systems.

The combination technique has already successfully been applied to linear initial value computations and eigenvalue computations in GENE [7,8]. One can observe that the execution times of the partial solutions do not solely depend on the size of the problem and the number of processors, but they are also severely affected by the anisotropy of the discretization. For a five-dimensional problem like GENE, where one has to deal with hundreds or thousands of partial solutions that differ in their discretization, this results in massive load imbalances. Therefore, in order to achieve full scalability on exascale systems, effective load balancing will be crucial. Previous approaches to distribute the computational load of the partial solutions [9] can only be successfully applied, if the computational effort is just coupled to the number of degrees of freedom and not the anisotropy of the grid.

1.1. Combination Technique

![Figure 1](image)

**Figure 1.** Grids of the classical combination technique for $d = 2$ and $n = 4$.

The sparse grid combination technique extrapolates the sparse grid solution by a superposition of partial solutions $f_l$ on coarse and anisotropic full grids $\Omega_l$ in the same physical domain. The discretization of each $\Omega_l$ is described by the corresponding level vector $\bar{l} = (l_1, \cdots, l_d)^T$, which refers to $2^{l_i} + 1$ degrees of freedom in the $i$th of $d$ dimen-
sions. All $\Omega_{\vec{l}}$, where $|\vec{l}|_1 = n + d - 1$ holds, are referred to have discretization level $n$. Combining the solutions $f_{\vec{l}}(\vec{x})$ by

$$f^{(c)}(\vec{x}) = \sum_{q=0}^{d-1} (-1)^q \binom{d-1}{q} \sum_{|\vec{l}|_1 = n + d - 1 - q} f_{\vec{l}}(\vec{x}),$$

retrieves a sparse grid approximation $f^{(c)}$ of $f$. See Figure 1 for a two-dimensional example. When we speak of a combination solution of level $n$ it means that the largest partial solutions used for the combination have discretization level $n$.

Throughout this paper we will focus on optimizing the scheduling of the partial solutions and thus the overall execution time of the combination technique. This will be the basis for the scheduling of advanced variants of the combination technique in the future.

1.2. GENE

GENE is a simulation code used for hot magnetized plasmas, which are for example occurring in devices for fusion energy research (tokamaks, stellarators) and astrophysical objects (e.g. solar flares). The underlying physical model simulated in GENE are the gyrokinetic equations, which are an adaption of the Vlasov-Boltzmann equation.

They are formulated in five dimensions: the spatial positions $x, y$ and $z$ of the center of gyration (guiding center) and the two velocity components $v_{\parallel}$, the velocity of the guiding center parallel to the magnetic field line, and $\mu$, which is the magnetic moment of the gyrating particle. For a generalized distribution function $g$, they can be formulated by

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

with $\mathcal{N}$ being a nonlinear operator on $g$ and $\mathcal{L}$ being a linear one (see [10,11] for details). Whereas the linear operator already governs the basic plasma behavior, only the nonlinear operator introduces the microturbulence into the system, which is the cause of turbulent transport.

Global nonlinear computations for computing the turbulent transport in current fusion experiments usually require a large amount of computational resources [1]. GENE is efficiently MPI parallelized, and it has been shown that it scales up to a few hundred thousand cores [12]. The parallelization is done by a domain decomposition in all five dimensions and for the different species (ions and electrons).

Even the simplest setup of a GENE simulation, namely the linear initial value runs, shows a large variance in the execution time for discretizations that have the same number of degrees of freedom and thus they will be studied in this paper. The chosen domain decomposition has probably the largest influence on the execution time. But with the automatic parallelization tool in GENE, it is chosen automatically and optimized for minimal execution time. Besides that, the execution time is influenced by the underlying hardware architecture and communication pattern. To cope with this, multiple kernels in GENE are implemented in a couple of different ways, which might lead to a better performance on a certain architecture. Throughout the autotuning phase, GENE can also
select the best performing kernels and expresses this in a so-called performance vector. Additionally, the behavior of the integro-differential operator $L$ is responsible for variance in the execution time. This comes from the fact that the number of integration steps depends on the resolution of the grid in only a subset of the five dimensions.

For the combination technique, grids with $2^l_i + 1$ grid points in each dimension are required. In contrast to that, GENE strictly requires equally sized subdomains for the parallelization. Thus, in order to obtain an efficient domain decomposition we have to use $2^l_i$ grid points for the calculations and interpolate the results to $2^l_i + 1$ grid points for the combination technique [7].

2. Proposed Load Model

In order to perform efficient scheduling, the runtime of the solution of all grids $\Omega_l$ contributing to the combination technique of a certain level $n$ have to be estimated. As mentioned in Sec. 1.2, several parameters affect the runtime of GENE. For the load model the anisotropy of the discretization and the number of degrees of freedom of the $\Omega_l$ are definitely the most critical parameters. Others, like the domain decomposition and the performance vector will not be part of the load model. The autotuning phase results in a well working parallelization and performance vector for each $\Omega_l$ and will therefore be used for production runs.

The other physical parameters of the simulation are taken from the GENE test suite, which provides a wide variety of parameter sets together with the expected output. Test case 1 of the suite is the linear initial value simulation of a single ITG instability and will be used throughout this paper, since it is robust and should be a baseline for further studies using more complex parameter setups. The linear initial value runs are completely decoupled in the $y$-dimension. Hence, it is sufficient to consider only four of the five dimensions for the combination solution.

Each measurement was performed with 32 MPI processes on a node of the supercomputer Hermit located at HLRS in Stuttgart, which consists of two AMD Interlagos processors with 16 cores each. For the given setup GENE requires a discretization level $l_i \geq 3$ in all dimensions. In the context of the combination technique only using partial solutions with $l_i \geq l_{\text{min}}$, where $l_{\text{min}} = 3$ in our case, is commonly referred to as a truncated combination [13]. With such a truncation the number of partial solutions is significantly reduced, but it also results in a worse accuracy of the approximation. Applying the truncation to the four-dimensional combination technique with $n = 17$ reduces the number of partial solutions from 9260 to 425. For this work, we were only interested in developing a strategy to cope with the load imbalances and did not consider any accuracy issues.

The only two variables used for the model are the number of unknowns of the partial solution $N := 2^{|l_i|}$ and the anisotropy of the corresponding grid $\Omega_l$. The anisotropy will be expressed by a vector $\hat{s}_l \in \mathbb{R}^d$. It is is defined as

$$s_{l,d} = \frac{l_d}{|l|_1}$$

for each grid $\Omega_l$. As $|\hat{s}_l|_1 = 1$ holds, a high value in one dimension will result in low values in at least one of the other dimensions. Therefore it is a good indicator for the
anisotropy of the discretization. For a perfectly isotropic grid it holds $s_{f,i} = \frac{1}{d}$. With this notation we can express the anisotropy of the grid completely decoupled from the number of grid points.

Our proposed load model has the form

$$t(N, \vec{s}) = r(N)h(\vec{s}).$$

(4)

The function $r(N)$ models the dependence of the execution time of a partial solution on the number of unknowns. The value provided by $r(N)$ is scaled by the function $h(\vec{s})$, which solely depends on the anisotropy of the discretization.

2.1. Modelling the Dependence on the Number of Grid Points

For the function $r(N)$ of the proposed load model (4) we need to choose a grid for each discretization level $n$, which serves as reference grid. A reasonable approach is to choose for every level the grid which has a certain anisotropy vector, like the isotropic grids with $s_{f,i} = \frac{1}{d}$. However, not for all $N$ exists a grid which exactly fulfills this condition. In these cases we choose the grid with $\vec{s}_f$ closest to the isotropic case (with respect to the $l_2$-Norm).

We choose our model

$$r(N) := mN^k + c,$$

(5)

with coefficients $m$, $k$ and $c$ to be fit to the measurement data in the least squares sense. The usual approximation of the dependence of the run time on the degrees of freedom for a simple explicitly solved initial value problem would be linear in $N$. But, since GENE uses an integro-differential operator with different discretization types in different dimensions, we chose a model $N^k$ since we assume the complexity to grow with a $k > 1$. The constant part $c$ is a term added to increase the accuracy of the model especially for the partial solutions with small $N$. For the larger grids, this will have basically no influence.

2.2. Modelling the Dependence on the Anisotropy

In order to create a model which solely depends on the anisotropy and not on the number of grid points, we normalize the runtime of each grid by the runtime of the corresponding isotropic reference grid

$$\tilde{t}(N, \vec{s}) = \frac{t(N, \vec{s})}{r(N)}.$$  

(6)

Experiments led us to the idea of using a polynomial ansatz for $h(\vec{s})$ of the form

$$h(\vec{s}) = c + \sum_{i=1}^{d-1} c_i s_{f,i} + \sum_{i=1}^{d-1} \sum_{j=1}^{d-1} c_{ij} s_{f,i} s_{f,j} + \sum_{i=1}^{d-1} \sum_{j=1}^{d-1} \sum_{k=1}^{d-1} c_{ijk} s_{f,i} s_{f,j} s_{f,k} + \cdots \approx \tilde{t}(N, \vec{s}).$$  

(7)
the different levels to fit the parameters of $h$ is not absolutely fair, since different number of measurement values were used for our assumption that the dependency of the execution time of a partial solution on the cases of different training data for the same degree of especially the most interesting values of the levels $14 \le n \le 17$. We repeated this procedure for different $n$ in order to see how the error changes when using different training data. We also examined the influence of our model on the accuracy of the model.

The coefficients of the polynomial are determined by fitting the model to the normalized measurement data $\tilde{r}(N, \tilde{s}_f)$. Our model is based on the assumption that the dependency on the anisotropy is decoupled from the number of unknowns $N$. This means that fitting the function $h(\tilde{s}_f)$ to the measurements taken for a specific $N$ should result in a good accuracy even when using $h(\tilde{s}_f)$ to estimate the execution times of partial solutions with a different $N$. This will be further discussed in Section 2.3.

### 2.3. Results of the Load Model

As it can be seen in Figure 2 (a), the function $r(N)$ covers the runtime of grids with larger $N$ quite well. It is desirable to achieve good accuracy in this range, since bad estimations of the largest grid result in high absolute errors for the load model. This is especially crucial for static scheduling. Thus we are not interested in increasing the accuracy for small grids at the prize of a more complex model. We used the measured run times of the eight (nearly) isotropic grids of discretization level $10 \le n \le 17$ to fit the parameters of $r(N)$ and obtained $m = 1.252 \cdot 10^5$, $k = 1.076$ and $c = 0.01308$.

In order to examine our assumption that the dependency of the execution time of a partial solution on the anisotropy is decoupled from the number of unknowns $N$, we performed the following experiment. We fitted the parameters of $h(\tilde{s}_f)$ by using only the measurements for grids of a specific level $n$ as training data. The function $r(N)$ was used with the parameters presented above. Then we evaluated the relative error $e$ of our model $r(N, \tilde{s}_f)$ with a set of test data which contained all the measurements from the partial solutions of levels $14 \le n \le 17$. We repeated this procedure for different $n$ in order to see how the error changes when using different training data. We also examined the influence of different degrees $q$ of $h(\tilde{s}_f)$ on the accuracy of the model.

The results are shown in Table 1. The case “all” refers to using all measurements of the levels $14 \le n \le 17$ as training data, which equals the whole test data set. Especially the most interesting values $e_{\sigma}$ and $e_{rms}$ vary only slightly when comparing the cases of different training data for the same degree of $h(\tilde{s}_f)$. This observation supports our assumption that the dependency of the execution time of a partial solution on the anisotropy is decoupled from the number of unknowns. Note, however, that the comparison is not absolutely fair, since different number of measurement values were used for the different levels to fit the parameters of $h(\tilde{s}_f)$ (see Table 2). Considering the degree of

### Table 1. Mean value $e_{\mu}$, standard deviation $e_{\sigma}$, maximum norm $||e||_{\infty}$ and root mean square $e_{rms}$ of the relative error $e$ of $r(N, \tilde{s}_f)$ for different degrees $q$ of the polynomial $h(\tilde{s}_f)$. Each row in the table refers to a different set of training data.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$q = 1$</th>
<th>$q = 2$</th>
<th>$q = 3$</th>
<th>$q = 1$</th>
<th>$q = 2$</th>
<th>$q = 3$</th>
<th>$q = 1$</th>
<th>$q = 2$</th>
<th>$q = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>0.0392</td>
<td>0.0582</td>
<td>0.0592</td>
<td>0.1533</td>
<td>0.0838</td>
<td>0.0945</td>
<td>0.4226</td>
<td>0.5382</td>
<td>0.6317</td>
</tr>
<tr>
<td>15</td>
<td>0.0229</td>
<td>0.0220</td>
<td>0.0207</td>
<td>0.1491</td>
<td>0.0793</td>
<td>0.0751</td>
<td>0.4032</td>
<td>0.5129</td>
<td>0.5319</td>
</tr>
<tr>
<td>16</td>
<td>-0.0027</td>
<td>-0.0212</td>
<td>-0.0222</td>
<td>0.1437</td>
<td>0.0732</td>
<td>0.0663</td>
<td>0.3871</td>
<td>0.3990</td>
<td>0.4097</td>
</tr>
<tr>
<td>17</td>
<td>0.0605</td>
<td>0.0253</td>
<td>0.0292</td>
<td>0.1546</td>
<td>0.0828</td>
<td>0.0784</td>
<td>0.4922</td>
<td>0.4244</td>
<td>0.3313</td>
</tr>
<tr>
<td>all</td>
<td>0.0326</td>
<td>0.0163</td>
<td>0.0151</td>
<td>0.1487</td>
<td>0.0763</td>
<td>0.0691</td>
<td>0.4382</td>
<td>0.4280</td>
<td>0.3839</td>
</tr>
</tbody>
</table>

Since it holds $|\tilde{x}_f|_1 = 1$, only three components of the four-dimensional $\tilde{x}_f$ can be chosen freely and thus the space of the polynomial reduces to three dimensions. We used polynomials of degree $q = 1, 2, 3$ for $h(\tilde{s}_f)$, which refers to cutting (7) after the second, third or fourth term respectively.
the polynomial, one can observe that $q = 2$ seems to be sufficient. Choosing $q = 1$ would lead to a significant increase of the error. With $q = 3$, however, the accuracy increased only slightly in some cases. The high $\|e\|_\infty$ are caused by outliers in the measurements, which appear seldom, but in non-deterministic fashion. We examined this behavior, but did not yet find an explanation. We expect it to be a peculiarity of GENE.

The coefficients of $h(\vec{s}, \vec{l})$ with $q = 2$ can be seen in Figure 2 (b) for different $n$. Since many coefficients are close to zero, an idea would be to only use the coefficients with the highest magnitude for the model. Thus already a small number of measurements would be sufficient as training data.

### 3. Scheduling

In order to exploit the two-level parallelism of the combination technique one might consider different options of distributing the partial solution among the available compute resources. In order to keep this approach as general as possible, in the following we will consider a partial solution a compute task, which will be processed by a process group. A process group is a subset of our available resources. For large-scale computations with GENE the resources would be a number of nodes on a modern HPC system. Thus a process group would consist of one or several nodes. Finding an appropriate division of the resources, however, is strongly dependent on the problem and the architecture of the system.

After one has decided on the division of the available resources into $p$ process groups, the next step is to distribute the compute tasks among the groups. Our goal is to minimize the overall time until all tasks are processed. This equals the time of the slowest group. In a more formal way, the optimization problem we have to solve is

$$\min_{\text{assignment}} \max_{1 \leq i \leq p} T_i$$

where $T_i = \sum_j t_{i,j}$ is the accumulated time of all tasks assigned to group $i$ and $t_{i,j}$ the time to solve task $j$ of group $i$. One could also incorporate the division of the resources into the optimization problem, which might lead to better results. As this would increase
the complexity of both, the scheduling algorithm and the load model, for now we only consider the case of equally sized process groups.

3.1. Static scheduling

Algorithm 1 represents a simple approach of distributing the tasks a priori based on the bin-packing algorithm first fit decreasing [14]. It is, however, not guaranteed to find the optimal solution to the problem formulated in (8) with such a greedy algorithm [14]. Considering, however, the fact that we are only solving the problem for the estimated execution times, it is a practicable approach, which yields acceptable results.

3.2. Dynamic Scheduling

Dynamic scheduling in the context of the combination technique can be performed in the following way: As in the static case we use a list with the estimated run times of the tasks sorted in decreasing order and accessible by all process groups. At the beginning one task from the head of the list is assigned to each process group. Whenever a process group finishes with its current task, it gets assigned the next task from the list. The advantage over the static algorithm is that wrong estimations will be compensated and thus the dynamic approach should yield better results.

4. Results

We compare our performance model to an early approach of performing load balancing for the combination technique presented in [9]. Their approach is based on the assumption that the execution times of the partial solutions scale linearly with the number of unknowns. With respect to the scheduling that means all partial solutions with the same discretization level $n$ have the same estimated execution time. This is a reasonable strategy when the problem is not sensitive to the anisotropy of the discretization or whenever no better model is available. Note that, in contrast to our approach, no measurement data is necessary. In the following, this method will be referred to as “linear model”.

Both approaches were applied to the static and dynamic scheduling algorithm presented in Section 3. Our test case was the four dimensional combination solution with level $n = 17$ of the linear initial value runs with GENE, as described in Section 2. The number of partial solutions that are required for the combination can be found in Table 2. We performed the scheduling experiments with different numbers $p$ of process groups. A process group corresponds to a full node on Hermit, as it was for the measurements (see Section 2). In order to calculate the overall time it takes until all tasks are processed, we
Table 2. Required number of partial solutions for combination solution of level $n = 17$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>56</td>
<td>84</td>
<td>120</td>
<td>165</td>
<td>425</td>
</tr>
</tbody>
</table>

Figure 3. Parallel efficiency $E_p$ over the number $p$ of process groups for the four different cases of scheduling. For static-linear and dynamic-linear the standard deviation is included.

used the real execution times from our measurements. With the linear model the overall run time strongly depends on the order in which tasks with identical estimated run times (all partial solutions of the same level) are placed in the list. Due to this dependence it would not yield very representative results to perform all experiments with the same order of the tasks (e.g. lexicographically ordered by the corresponding level vector). Therefore, before scheduling the tasks by the estimated run time, we randomly permuted the tasks of each level. We repeated this procedure 1000 times for each $p$ and computed mean and standard deviation.

The results for all four combinations of scheduling algorithm and load model can be found in Figure 3. As a meaningful metric to compare results for different $p$, we used the parallel efficiency

$$E_p = \frac{T_1}{pT_p},$$

where $T_1$ is the time it takes to compute all tasks with only one process group and $T_p$ the time with $p$ process groups, respectively. For dynamic scheduling with the anisotropy model (dynamic-anisotropy) a parallel efficiency of more than 97% can be observed for up to $p = 113$ before the efficiency drops significantly. Performing dynamic scheduling with the linear model (dynamic-linear) the performance is similar to the dynamic-anisotropy case up for to $p = 50$ before the efficiency drops. The largest difference of the mean value to the dynamic-anisotropy case is 14% for $p = 110$. For $p = 165$ the efficiency has converged to the value of the dynamic-anisotropy case. At this point, $p$ equals the number of partial solutions of level 17, which means that each process group will start with one of the largest tasks. Due to the dynamic assignment of the task whenever a process group has finished with the current task, the final distribution of the tasks
on the process groups is very similar to the dynamic-anisotropy case, although different models to estimate the run times have been used. For both cases of static scheduling the efficiency is significantly lower than in the dynamic cases. For static scheduling with the anisotropy model (static-anisotropy) the largest difference to dynamic-anisotropy was 22% at $p = 110$. It converges to the value of the dynamic-anisotropy case for $p = 157$. For static scheduling with the linear model (static-linear) the largest difference to the dynamic-anisotropy case was 34% for $p = 110$. To the static-anisotropy case the largest difference was 14% at $p = 157$.

5. Summary and Outlook

At the example of linear initial value runs in GENE, we introduced a load model for the combination technique which is capable of estimating the execution time of a partial solution based on the number of unknowns and the anisotropy of the discretization. Note that the model is not only applicable to GENE, but it is meant to provide a general load model for the combination technique. The model was compared to a standard approach, which is solely based on the number of unknowns, by performing experiments with static and dynamic scheduling. In both cases, we have achieved a significant increase in parallel efficiency over the standard model.

Further work will include the development of a strategy to select the training data for the model in such a way that only a minimal number of data is required. It would also be desirable to gather the training data at run time, so that no additional measurements are necessary. This could be done by starting the computation with the tasks which have been identified to be most significant to fit the model’s parameters.

Acknowledgements

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

References


