DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Interdisciplinary Project

Implementation of the Dimension-Adaptive Combination Technique into a Parallel Framework

Author:	Simon Griebel
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Michael Obersteiner
Submission Date:	04.01.2019

Abstract

The so called (sparse grid) combination technique is an elegant way of combining the properties of sparse grids, while still being able to use PDE solvers, that support only standard full grids. Since there are a lot of possible combinations (which don't necessarily have to resemble a sparse grid), it is possible to adapt the combination at runtime to generate more optimal results for each use case, eliminating the need for manual optimization.

The focus of this project was to implement such an adaptive algorithm in a framework that only supported the static combination technique. The chosen refining criterion was based on the difference between the solution of the PDE on two different grids that are used in the combination. The resulting algorithm was tested with the plasma physics framework GENE. Due to possibly remaining bugs only basic properties of the implemented algorithm were examined.

Contents

1. Introduction

Equidistant full grids are the simplest way to discretize the spatial dimension of PDEs. The problem is, that they suffer from the curse of dimensionality. The number of needed grid points increases exponentially in the number of dimensions, if the resolution of the grid stays the same. There are many different approaches to reduce the number of needed points, but especially for higher dimensions sparse grids seem to be a promising approach. Their main disadvantage is the more complex implementation and so most of today's PDE solvers don't support them. The so called combination technique offers compatibility with existing solvers, while also preserving the properties of sparse grids (in certain cases). Here multiple solutions of the problem on differently sized full grids can be combined to resemble a sparse grid solution, but the technique is much more flexible. Many different combinations are possible and to be able to fully exploit this method, it makes sense to think about algorithms that optimize the combination at runtime to achieve the best result possible.

The focus of this project was to implement such an adaptive algorithm into a framework that is based on the works of [1] and that only supported the static combination technique. The implementation was then tested with the plasma physics framework GENE [2]??.

This paper will firstly introduce the theoretical background/motivation for using sparse grids and the (adaptive) combination technique. Then it will go in to the details of implementing such an adaptive algorithm into an existing framework. The paper then concludes with the results that were achieved with this implementation on the plasma physics framework GENE followed by an outlook on possible expansions of the implementation and a conclusion.

2. Theoretical background

2.1. Sparse Grids

As mentioned sparse grids, which are based on the work of [3] and which are described in more detail in [4], try to improve on equidistant full grids by using less grid points while achieving a similar accuracy. For this paper only full grids that have $2^{n_i} \pm 1$, $n_i \in \mathbf{N}_+, i \in [d]$ grid points in dimension i are taken into account[1], where d is the dimension of the problem to be solved. If it is assumed, that there are the same number of points $2^n \pm 1$ in each dimension then the number of points has the complexity $O(2^{nd})$. A comparable sparse grid needs only $O(n^{d-1}2^n)$ points, while the results are generally only slightly worse than on full grids. For a function that has a bounded second derivative the error is only $O(2^{-2n}n^{d-1})$ compared to $O(2^{-2n})$ in a full grid (in the L_2 norm). The problem is that sparse grids are a lot more complicated to implement than full grids and so most PDE solvers don't support them. Their structure is more complex and they work on a hierarchical basis.

2.2. Combination Technique

The so called combination technique, which is described in more detail in [5] is a way to get the complexities of sparse grids (both for the error and the number of points), while still being compatible with full grid solvers. Here the result of the computations on different full grids can be combined to approximate the result on a sparse grid. For that we use the same full grids as described in the previous section. To combine the result of a calculation on two full grids, one adds the values of grid points that exists in both, and simply takes the values for points that exists only in one of the two grids. Since the values of the points that are in both grids are sort of counted twice, one has to subtract a third grid, which is made up of all grid points that are contained in both grids. This approach can be generalized for a combination of arbitrarily many grids, where it gets much harder to see which subgrids have to be subtracted to correct the combination. It can happen that some subgrids have to be subtracted more than once and it can also happen that some subgrids (of the initially subtracted subgrids) were

[1] $2^{n_i} + 1$ points if there are values on the boundary, otherwise $2^{n_i} - 1$

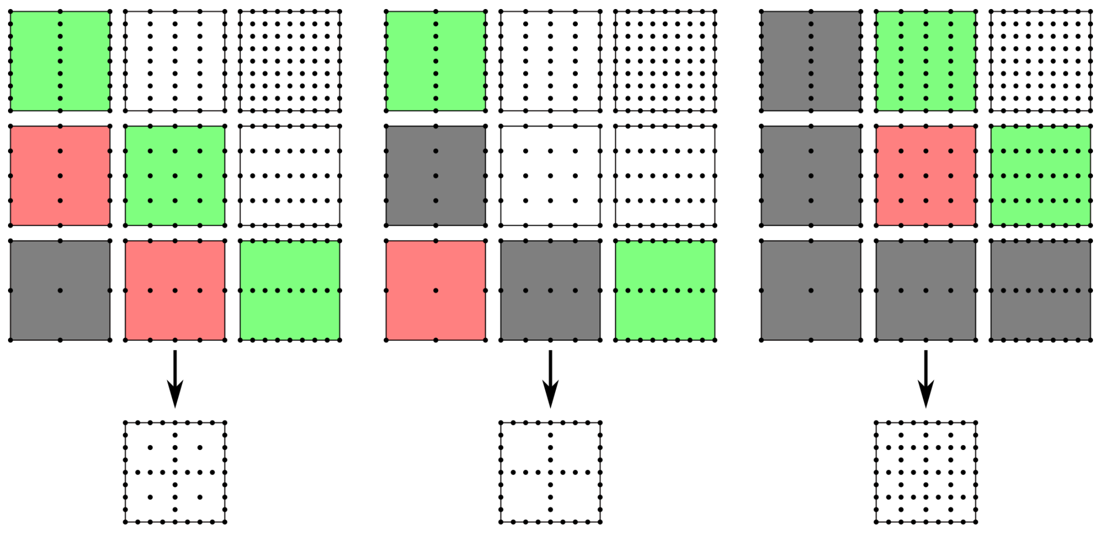

Figure 2.1.: Three possibilities of combining full grid solutions with the combination technique. On the left the result is the standard sparse grid. The results of the green grids are added together and the red grids are subtracted once to get the grids on the bottom. The grey grids are the ones contained in the resulting combined grid without taking part in the calculation.

subtracted too many times and have to be readded again. Both of these effects only start to appear in dimensions $d \geq 3$.

There is of course a way to mathematically formalize this approach. For that we use the same full grids as in the previous section, that are determined by their index vector $\vec{n} \in \mathbb{N}_+{}^d$. The grids that should be combined will from now on be called active grids. These are the grids that are not contained by another grid in the combination as a subgrid and they unambiguously identify a certain combination, since all of their subgrids are contained in the combination by definition (even if they are not directly needed for the calculations). A grid $\vec{m}$ is a subgrid of grid $\vec{n}$, if $\vec{m} \leq \vec{n}$ (and $\vec{m} \neq \vec{n}$). Let I be the set of the index vectors of all grids contained in the combination, then the solution f^{ct} of the combination can be described by the linear combination of the full grid solutions $f_{\vec{n}}$ on grid $\vec{n}$ as follows:

$$f^{ct} = \sum_{\vec{n} \in I} c_{\vec{n}} f_{\vec{n}} \tag{2.1}$$

The coefficient $c_{\vec{n}}$ denotes how many times each grid $\vec{n}$ has to be added or subtracted. With χ^I being the characteristic function of the set I [2], it can be calculated as follows [6,

[2] $\chi^I(\vec{x}) = 1$, if $\vec{x} \in I$, else $\chi^I(\vec{x}) = 0$

p. 114]:

$$c_{\vec{n}} = \sum_{\vec{z}=\mathbf{0}}^{\mathbf{1}} (-1)^{|\vec{z}|_1} \chi^I(\vec{n} + \vec{z}) \tag{2.2}$$

If a coefficient is 0, no solution has to be calculated for the corresponding grid. Grids that are not active and have a coefficient that is non-zero will be called correction grids from now on.

Three possible grid combinations are shown in Figure 2.1. As can be seen one can generate a sparse grid with this approach, but not only that.

It has to be mentioned that combining different full to a sparse grid is generally not the same as doing the calculations on the sparse grids directly, but in [5] it was shown that under certain assumptions the error of the combination has the same complexity as the error on the sparse grid.

Since there are a lot of possible, feasible combinations and because it is generally not obvious for a given problem, which combination might lead to the best results, it makes sense to think about an algorithm that automatically tries to find the best combination. This approach is described in the next section.

2.3. Adaptive Combination Technique

2.3.1. Basics

There are two main things that have to be considered when making the combination technique adaptive. The general expansion strategy has to be chosen, which determines which grids can be added or removed. Then an error/refining measure has to be chosen, which decides which of the candidates (if any) is chosen. The general strategy that is used in this paper is the same as described in [7].

All of the grids that are contained in a combination are divided into an active set A and an old set O. The active set contains all active grids, which are colored green in all figures in this paper. The old set consists of all correction grids, which are painted red, and all other contained grids, which are painted grey.

In each expansion step the error measure is calculated for each grid in the active set. If a grid is chosen for expansion, all forward neighbors, whose backward neighbors are all in the old set, are added to the active set. The grid chosen for expansion is removed from the active set, even if not all forward neighbors could be added. The complete expansion step is summarized in Algorithm 1 and the expansion process is illustrated in Figure 2.2. It is important to note that not all possible combinations can be generated with this technique as illustrated in Figure 2.3.

```
choose $\vec{n} \in A$ according to the error measure
$O = O \cup \vec{n}$
$A = A \setminus \vec{n}$
for $t = 1, ..., d$ do
    $\vec{m} = \vec{n} + \vec{e_t}$
    if $m_u = 1 \vee \vec{m} - \vec{e_u} \in O$ for all $u = 1, ..., d$ then
        $A = A \cup \vec{m}$
    end
end
```

Algorithm 1: Basic expansion step. Slightly modified version of the algorithm presented in [7], that highlights that grids at the border don't have a backward neighbor in that direction.

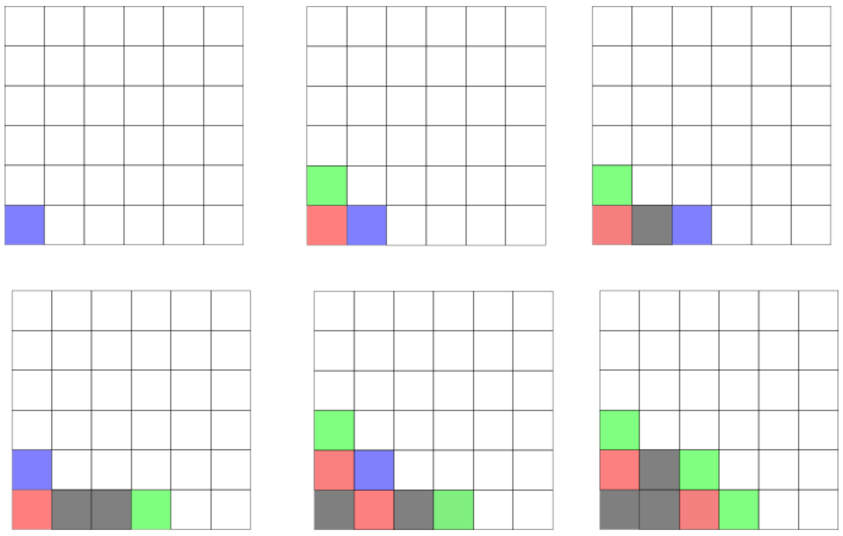

Figure 2.2.: Possible expansions steps starting from a single grid. The green and red grids have the same meaning as in figure 2.1 and the blue grid is the grid that is chosen for expansion in each step.

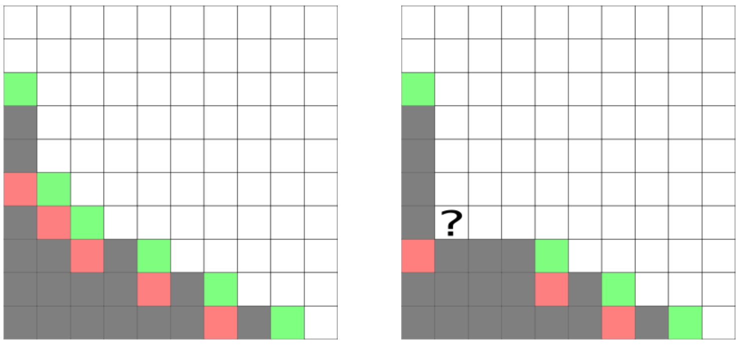

Figure 2.3.: On the left there is an example for a possible result of the expansion algorithm starting with only a singled grid. On the right there is an impossible result. At least the grid marked with the '?' would have to be contained in the combination for it to be valid, because it would be impossible to add it (and all grids with a bigger index vector) with the given algorithm. The middle example of Figure 2.1 would also be impossible.

3. Implementation of the Adaptive Combination Technique

3.1. Error Measure

For the implementation of the error measure as part of this project an approach was chosen that compares the values of the solution on an active grid $\vec{a}$ to a correction grid $\vec{c}$ that is a subgrid of it. The reason for including the correction grids in the error measure is that they have a lower resolution then the active grids, so it is expected that the solutions calculated on these grids are worse than on the active grids, which might badly influence the combined solution.

Let $f_{\vec{a}}(x)$ be the solution of the problem for a point x in the active grid and $f_{\vec{c}}(x)$ the same for a correction grid. Let $X_{\vec{c}}$ be the set of points in the grid $\vec{c}$ (which are are contained in $\vec{a}$ as well), then the error measure $\epsilon_{\vec{a}}$ can be calculated as follows:

$$\epsilon_{\vec{a}} = \max_{x \in X_{\vec{c}}} \frac{|f_{\vec{a}}(x) - f_{\vec{c}}(x)|}{\max_{x \in X_{\vec{c}}} |f_{\vec{a}}(x)|} \prod_{i=1}^{d} \frac{1}{2^{a_i}} \tag{3.1}$$

The basis of this error measure is the pointwise difference between the two solutions. This difference is then divided by the maximum value of the solution on grid $\vec{a}$ to keep small, unimportant fluctuations around 0 from completely dominating the error measure. This is then again divided by (roughly) the number of points in $\vec{a}$ to favor less computationally expensive expansions. The choice that the final error measure is the maximum of these values and not something like the (squared) mean is purely arbitrary and not based on empirical evidence.

For each active grid the partner needed for the error measure is chosen by decreasing one index of $\vec{a}$ until such a grid is hit. This is tried for every dimension until a partner is found (always restarting from $\vec{a}$). This means that there must be at least one such partner for every active node. As it turns out, this is always the case, when the general expansion strategy described in the previous section is used. The only thing that has to be ensured, is that one doesn't start with a single grid in the combination, but starting with at least one additional grid in each dimension is already enough (in 2D this corresponds to the left image in Figure 2.2).

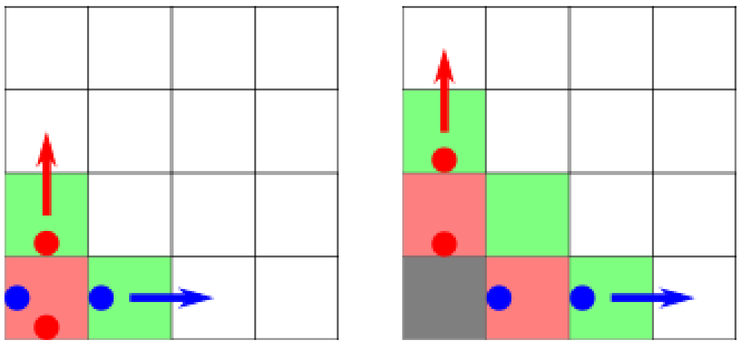

Figure 3.1.: On the left there is minimal starting grid for the used expansion strategy, which ensures that every valid combination in 2D can be generated with the given expansion rules. On the right there is the minimal starting grid for the alternative expansion strategy. The possible expansions are marked with an arrow. The circles in the corresponding color denote the grid pairs that are used for calculating the error measure for the expansion.

Another expansion strategy was tried out shortly, but due to certain problems it was not fully implemented as part of this work. Its ideas and problems are described in Section 3.3.

3.1.1. Theoretical Problems

There are some fundamental problems with this form of adaptivity. One is rooted deeply in the definition of the combination technique that is used here. Each grid that gets added in the expansion has roughly twice as many points as its backward neighbor. This means one can only add so many grids in a certain direction before the runtime increases too much. This very variable runtime is especially problematic for clusters where you have to specify the runtime of your algorithm beforehand. Either you have to specify too many processes in the beginning which leads to idle processes or you specify to few, than it will take much longer to finish. It is of course possible to set a maximum number of grid points, that will be added, but that heavily reduces the flexibility of the algorithm.

Then there is a problem that is connected to the applicability of this adaptive scheme to PDEs (though it might also hold true for other use cases). It is unlikely for complex PDEs to reach a sensible result on very coarse grids. That also means that the error measure may not be very useful in this case, which leads to random looking expansions, which may not even improve the convergence behavior of the PDE. It may even be impossible for a certain PDE to converge with the given adaptive technique, since there will always be grids directly at the boundary, which have at least on dimension, in which they only contain 2 ± 1 points and so for certain PDEs they might ruin the result of the complete combination. For the last problem there is at least a static solution. It is possible to introduce a 'minimum' vector $\vec{b} \in \mathbb{N}_+{}^d$. Now everything works as before, but each grid $\vec{n} \in \mathbb{N}_+{}^d$ now has $2^{n_i+b_i} \pm 1$ points, which avoids grid that are to coarse in certain dimensions, but this heavily reduces the flexibility of this approach.

In the framework that is presented in the next section this was implemented by explicitly increasing the index vectors, so using vectors $\vec{m} := \vec{n} + \vec{b}$ (so $m_i \in \mathbb{N}_+ \setminus [b_i], i \in [d]$) since this approach was more natural to implement with the already existing interfaces. It has the same effects, but some algorithm changes are now required. The minimal starting grid is now $\vec{b} + \mathbf{1}$ (instead of $\mathbf{1}$) and in Algorithm 1 the check $m_u = 1$ has to be changed to $m_u = b_u + 1$.

3.2. Concrete Implementation

3.2.1. Description of the Existing Framework

The presented algorithm was implemented into the code framework created as part of [1]. The algorithm was integrated into the existing main program loop that is described in Algorithm 2 (in its current form). In the loop a global master is responsible to distribute the workload, consisting of the active and the correction grids, among different process groups. Each group consists of a (globally) fixed number of processes, that together calculate the solution for a given grid. The master communicates only with one of these processes, the group manager. This manager splits the workload between itself and all other processes in the group. This is done by actually splitting the given grids, so each group member calculates a part of each grid that is given to the group.

To avoid a divergence of the solution on the differently sized grids after some iterations, the full grid solutions are regularly hierarchized and then combined to the combination solution (which is distributed among all groups). This solution is then used to reset the values on all fullgrids, where the values are then dehierarchized again. For all simulations in this paper this was done every iteration. This approach has the big advantage, that even grids that were not part of the combination can be populated with the values from the combined solution, which made the implementation of the expansion a lot simpler.

```
*Distribute the grids among the processes
*Distribute the grid-group-map
*Initialize all new grids with the combined solution
 Calculate the solution on each grid
″Calculate the error measure for each grid and determine the best
 Determine the combined solution
″Update the combination scheme with the chosen expansions
″Remove all fullgrids from all processes
```

Algorithm 2: The main program loop. Steps marked with a * only need to be executed when an expansion was added in the previous step. Steps that are marked with ″ only need to be executed when an expansions should happen in the current step.

3.2.2. Implementation of the Expansion Algorithm

The described process structure makes the implementation of the adaptivity a bit more complex since each group worker can only calculate the error information for its part of each grid. This information than has to be combined in the manager. This is done for all grids the group owns. The manager then sends the index vector and the error measure of the grid with the biggest error measure to the master, which then determines the best grids of all groups. The chosen grid is then sent back to the managers, which in turn send it to the workers. All processes (master, manager, workers) then update their local copy of the state of the combination. After such an update all grids are redistributed by the global master.

The described process structure also makes the calculation of the error measures on the workers quite difficult. As mentioned in the previous section, two grids are needed for that, but it is impossible to guarantee that both of them are evaluated by the same group, if duplicate calculations are to be avoided. So while distributing the workload among the different process groups, the master creates a map, which maps each grid to the process group that owns it. After the distribution is completed and before the calculations are started the master sends this map to every group, which in turn distribute it among their members. Since each group member knows the current state of the combination, they can now calculate which grids are needed for the error measure calculations and look up by which group they are owned. Now comes an important implementation detail: The grids are split between the group members the same way in every group. That means that if a point x is contained in grid g_1 and was given to the process with offset i in the group a, then it is guaranteed that each grid g_2 that also contains x has to have the same offset i in its group b. This means every group member, by knowing its own offset in its group, can now calculate the error measure for its part of each grid.

So after the master now sends the signal to each group to start the error measure calculation, each group member iterates through every active grid and checks if the group either owns that grid or the correction grid needed for the error measure calculations. If the group owns both, all of its processes can simply do the calculations on their own. If it owns the active node, then its processes wait to receive the information of the group owning the partner grid, which in turn sends it. If neither the active grid nor the correction grid is owned by a group, nothing has to be done for that active grid.

The loop to calculate the error measures is described in Algorithm 3.

```
foreach $a \in A$ do
    Calculate the error measure partner $c$
    $aGroupID = gridGroupMap[a]$
    $cGroupID = gridGroupMap[c]$
    if $myGroupID = aGroupID$ then
        if $myGroupID \neq cGroupID$ then
            receive point data of $c$ from $cGroupID + myoffset$
        end
        calculate local error measure
        the group manager combines the local error measures of its workers
        the group manager updates the currently best error measure
    end
    else if $myGroupID = cGroupID$ then
        send point data of $c$ to $aGroupID + myoffset$
    end
end
the manager sends the best error measure to the master
```

Algorithm 3: The loop that is executed on each process (except the global master) when the error measure should be calculated.

This whole algorithm may be inefficient but due to the fact that there will generally be a lot less expansion steps compared to total steps of the simulation the overhead due to a non optimal algorithm should be negligible.

3.3. Failed Expansion Strategy

The first idea for an expansion strategy was to add at most one grid in every expansion step, while using the same error measure as above. This is highly problematic, since it now becomes necessary to add a grid that is not a neighbor of an active (or a correction) grid. To be exact one has to iterate over all grids, where at least one forward neighbor is not already in the combination scheme. To check whether an expansion is valid or not one uses the same criteria as in the used expansion strategy. Since the grid $\vec{n}$, that has a possible expansion as neighbor, may not be directly involved in the calculations, there aren't any values directly associated with it, with which the error measure can be calculated. Nevertheless $\vec{n}$ is part of the combination, so by definition its points are all contained in at least one active grid. So to get an error measure for this expansion one needs to first search for at least one of these active grids. This is done by repeatedly increasing the index in a certain dimension by one until an active grid is hit. This dimension must be orthogonal to the expansion direction.

For this approach to work the minimal starting combination scheme has to be bigger than in the used expansion strategy as illustrated in Figure 3.1. The reason is that it would be impossible to ever get a grid, which doesn't lie directly on the boundary in the simpler starting position. Figure 3.2 shows a more advanced combination scheme with this technique and illustrates that the grids used to determine the error measure for an expansion might be far away from the expansion itself.

The main reason why this algorithm was not implemented is mainly its implementation complexity, while it is not obvious why it should be any better than the implemented algorithm. Here it takes even more expansion steps until a correction grid, that might lead to high error measures is replaced by correction grids with higher resolution.

The main advantage of this approach would be that more combination schemes are now reachable because the expansion is more fine grained.

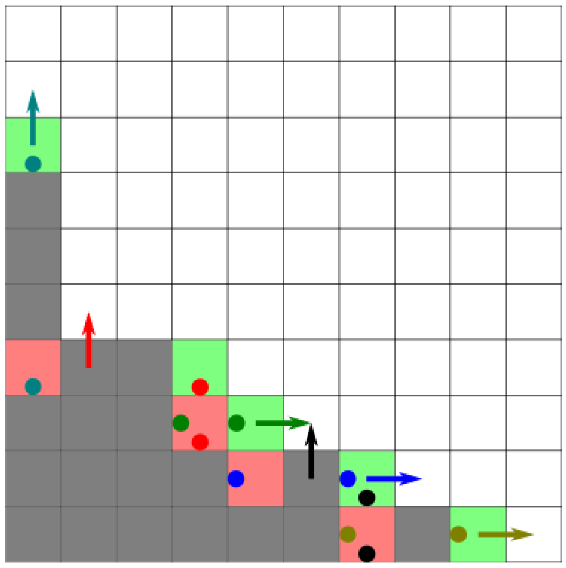

Figure 3.2.: Illustration of the failed expansion strategy. The expansion indicated by the red and black arrows show the disconnection between the grids used to determine the error measure and the grid that will be added if the expansion in question is chosen. For an explanation of the symbols see Figure 3.1.

4. Results

4.1. General

As mentioned the examples were tested with GENE [2], which deals with plasma physics and the inner workings of which are described in more detail in [8], as underlying solver. While the solver uses six dimensions there are only three in which an expansion makes sense, which is why all grid index vectors will be three dimensional.

Due to the complexity in the given code framework, it took a lot longer than expected to implement the previously described algorithm. For that reason there was not enough time to really test the implementation for remaining bugs and to tweak it for better results.

The parameter file that was used as a basis for all tests can be found in the Appendix A. Values that were changed for the tests are explicitly mentioned.

4.2. Test Scenarios

Since each combination is defined by its active grids, the state of a combination will be described by a list of the indices of all active grids.

The reference for the correctness of the result was the fullgrid [6, 6, 6] (which is denoted in the parameter file by $lmin = 2\ 1\ 6\ 6\ 6\ 1$ and by $lmax = 2\ 1\ 6\ 6\ 6\ 1$). These results were then transferred to a [7, 7, 7] full grid through the hierarchization-combination-dehierarchization-process mentioned in Section 3.2.1. This transfer was done for all tests, where results were compared, to create common baseline.

Two different starting combinations were used for the expansion tests. The first one is [5, 4, 4], [4, 5, 4], [4, 4, 5] (which is denoted in the parameter file by $lmin = 2\,1\,4\,4\,4\,1$ and by $lmax = 2\ 1\ 5\ 5\ 5\ 1$), which will be called 444-scenario from now. With this starting combination five test runs were made, where an expansion was done every x iterations with $x \in 250, 300, 400, 500, 600$. The second starting configuration was $[6, 5, 5], [5, 6, 5], [5, 5, 6]$ (555-scenario). Here only one test run was made with an expansion every 1200 iterations. The last test was made, because a test with $lmin = 2\,1\,4\,4\,4\,1$ and $lmax = 2\,1\,7\,7\,7\,1$ with the static combination technique did not converge to the reference solution of the full grid (with an relative mean square error

a	b	#Groups	#Procs. per Group	Exp. 1	Exp. 2	Exp. 3
4	6	4	128	[6,4,4] 1.3	[7,4,4] 2.7	[5,4,5] **22**
4	7	4	128	[7,4,4] 2.3	[8,4,4] **42**	[6,4,5] **65**
4	8	4	128	[9,4,4] **209**	—	—
5	6	4	128	[6,5,5] 5.0	[7,5,5] 5.2	[8,5,5] 6.7
5	6	4	64	[6,5,5] 5.2	[7,5,5] 6.1	[8,5,5] 7.0
5	6	2	128	[6,5,5] 5.1	[7,5,5] 5.1	[8,5,5] 6.7
6	7	4	128	[7,6,6] 9.7	[8,6,6] 13	—

Table 4.1.: The runtime of the expansion algorithm for different configurations. The Exp. columns denote the active grid that was chosen for the expansion and the runtime of the expansion algorithm in seconds.

of roughly 140%), while a static run with $lmin = 2\ 1\ 5\ 5\ 5\ 1$ and by $lmax = 2\ 1\ 7\ 7\ 7\ 1$ did converge (with a relative MSE of only 7.0%).

4.3. Runtime of the Expansion algorithm

It is expected that the runtime of the expansion algorithm itself scales roughly linearly in the number of active grids and the number of points per active grid, but due to the unpredictability in the distribution of the grids to the processes there is a big random factor. These measurements exclude the filling of the new grids with values but they include some folder operations that are necessary for the interface between this framework and gene.

In the 555-scenario the measured runtime of all 4 expansion steps together was only $0.76s$ which seems quite low, but due to the expansion that only ever went in one direction (which can be seen in the next section) the amount of active grids stayed constant at 3.

Instead of the 444-scenarios some smaller special scenarios were used to test the runtime. All of them ran only for up to 50 iterations, with up to 3 expansion steps. Here the number of processes and the starting configurations varied, but the configurations were always of the form $lmin = 2\ 1\ a\ a\ a\ 1$ and by $lmax = 2\ 1\ b\ b\ b\ 1$. The scenarios that used 128 processes per group used $p = 1\ 1\ 1\ 8\ 16\ 1$ while the one scenario that used 64 used $p = 1\ 1\ 1\ 8\ 8\ 1$. The results are listed in Table 4.1.

The most interesting measurements are bold. It is not clear why these take so long for their expansion steps. It can't be due to the increasing number of active nodes, since none of these expansions increase this number. One possibility would be a very suboptimal redistribution of the grids among the processes but it is questionable, if this

Active Grid	[5,4,4]	[6,4,4]	[7,4,4]	[8,4,4]	[4,5,4]	[9,4,4]
Error Measure	$1.5 \cdot 10^{-5}$	$4.2 \cdot 10^{-4}$	$1.5 \cdot 10^{-4}$	$6.2 \cdot 10^{-5}$	$4.1 \cdot 10^{-5}$	$3.1 \cdot 10^{-5}$
Active Grid	[5,5,4]	[4,4,5]	[5,4,5]	[6,4,5]	[7,4,5]	[8,4,5]
Error Measure	$2.9 \cdot 10^{-5}$	$5.0 \cdot 10^{-4}$	$4.9 \cdot 10^{-5}$	$5.1 \cdot 10^{-5}$	$5.5 \cdot 10^{-4}$	$3.5 \cdot 10^{-5}$
Active Grid	[6,5,4]	[7,5,4]				
Error Measure	$3.2 \cdot 10^{-5}$	$5.1 \cdot 10^{-4}$				

Table 4.2.: Active grids that were chosen for expansion in the 444-scenario with $x = 400$ (in order left to right then top to bottom)

Active Grid	[6,5,5]	[7,5,5]	[8,5,5]	[9,5,5]
Error Measure	$1.22 \cdot 10^{-6}$	$3.2 \cdot 10^{-6}$	$1.9 \cdot 10^{-5}$	$7.6 \cdot 10^{-6}$

Table 4.3.: Chosen expansion grids for the 555-scenario (in order left to right then top to bottom)

can have such a big impact in these cases. Further testing has to be done, whether this can be consistently reproduced or if these are just inconsistent measurement blips.

Except for the $209s$ these costs are still negligible compared to the runtime of the complete simulations, especially if one considers that the number of expansion is always far less than the number of iterations.

4.4. Expansion Patterns

For the 444-scenario with $x = 400$ the expansion represented in Table 4.2. As can be seen the expansion was heavily skewed towards the first dimension. This was not justified since the solution did not converge to the full grid reference (again with a relative MSE of 140%). This convergence and expansion behavior was the same for all 444-scenarios and as can be seen in Table 4.3 at least the expansion behavior of the 555-scenario was also similar. Due to the long running time ($> 30h$) of the latter scenario no convergence was tested here. A very small indication for the correctness of the calculation of the error measure is that for the 555-scenario the calculated error measures are generally lower than in the 444-example.

5. Outlook

Due to the slow implementation progress a lot can still be done. The first thing that comes to mind is that the algorithm has to be verified more rigorously. It may be helpful to create simple examples, where optimal expansion strategies calculateable by a human. After the correctness of the algorithm is verified (at least with a somewhat high probability) it should be tried out on more examples with different configurations (different starting grids, different amount of processes) and on different solvers, since one solver is not enough to get a good idea of the performance of the presented approach.

Then there are a lot of tweaks that can be made to te algorithm. Firstly the error measure calculation can be changed by trying out different norms, using the average error instead of the maximum, etc. The error measure can also be calculated on the hierarchical basis to see if it makes a difference.

As mentioned in previous sections the algorithm has some fundamental flaws which hinder its flexibility and effectiveness. One of them was that too coarse correction grids are most likely the reason for a high error measures but the algorithm does not directly increase the resolution of these grids, but only of the active grids, which already have a higher resolution. Only after some expansions the correction grids tend to increase their resolutions as well. It would be interesting to see if it is possible to implement an efficient expansion algorithm that directly increases the resolution of the correction grids.

Another big change would be to make the expansion strategy more flexible so that more combination schemes are legal. This is connected to the idea that it would be nice for the algorithm to realize on its own that it mustn't allow grids that are to coarse in certain directions to be involved in the calculation, which is now done by setting the minimum index vector as described in Section 3.1.1.

6. Conclusion

The results of the implemented algorithm were somewhat underwhelming. While the runtime of the expansion algorithm itself was ok, the expansion patterns were not, but this has more to do with the low amount of time that was invested to investigate possible reasons. Additionally, for every fundamental problem the current algorithm has there where some ideas of how to fix them or at least lessen their effect. So in the end the results achieved in this project are only the foundation for further research that explores the possibilities of using the adaptive combination technique for PDEs with the presented framework.

A. Basic Parameter file

```
[ct]
#last element has to be 1 -> specify species with special field
dim = 6
lmin = 2 1 5 5 5 1
lmax = 2 1 6 6 6 1
leval = 2 1 4 4 4 1
leval2 = 2 1 7 7 7 1
p = 1 1 1 4 8 1
ncombi = 6000
readspaces = 1
fg_file_path = ../plot.dat
fg_file_path2 = ../plot2.dat
boundary = 1 0 1 1 1 0
hierarchization_dims = 0 0 1 1 1 0
reduceCombinationDimsLmin = 0 0 0 0 0 0
reduceCombinationDimsLmax = 0 0 0 0 0 0

[application]
dt = 0.005
combitime = 10000
nsteps = 1
shat = 0.7960
kymin = 0.3000
lx = 4.18760
numspecies = 1
GENE_local = T
GENE_nonlinear = F

[preproc]
basename = ginstance
executable = ./gene_new_machine
mpi = mpirun
```

```
sgpplib = [...]/combi/lib/sgpp
tasklib = [...]/combi/distributedcombigrid/examples/gene_distributed/lib
startscript = start.bat

[manager]
ngroup = 4
nprocs = 32

[faults]
[deactivated]
```

Bibliography

[1] M. Heene. *A massively parallel combination technique for the solution of high-dimensional PDEs*. Stuttgart, 2017.

[2] *The GENE Code*. URL: `https://genecode.org/`.

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

[4] H.-J. Bungartz and M. Griebel. "Sparse grids." In: *Acta Numerica* 13 (2004), pp. 147–269. DOI: `10.1017/S0962492904000182`.

[5] M. Griebel, M. Schneider, and C. Zenger. "A Combination Technique For The Solution Of Sparse Grid Problems." In: *Iterative Methods in Linear Algebra* (Oct. 2000).

[6] A. Rüttgers. *Multiscale Simulation of Polymeric Fluids using Sparse Grids*. Bonn, 2016.

[7] J. Garcke. "A dimension adaptive sparse grid combination technique for machine learning." In: *Proceedings of the 13th Biennial Computational Techniques and Applications Conference, CTAC-2006, Volume 48 of ANZIAM J.* 48 (Jan. 2007).

[8] W. Dorland, F. Jenko, M. Kotschenreuther, and B. Rogers. "Electron Temperature Gradient Turbulence." In: *Physical review letters* 85 (Jan. 2001), pp. 5579–82. DOI: `10.1103/PhysRevLett.85.5579`.