DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Engineering Science

Implementation of a Parallel Sparse Grid Combination Technique for Variable Process Group Sizes

Martin Molzer

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Engineering Science

Implementation of a Parallel Sparse Grid Combination Technique for Variable Process Group Sizes

Implementierung der parallelen Sparse Grid Kombinationstechnik für variable Prozessgruppengrößen

Author:	Martin Molzer
Supervisor:	Prof. Dr. rer. nat. habil. Hans-Joachim Bungartz
Advisor:	M.Sc. Michael Obersteiner
Submission Date:	15.01.2018

I confirm that this bachelor's thesis in engineering science is my own work and I have documented all sources and material used.

Munich, 15.01.2018 Martin Molzer

Acknowledgments

I want to acknowledge,

. . . my mother and family for their unconditional patience and support.

. . . my advisor, Michael Obersteiner, for his quick and accurate replies to questions.

Abstract

In high dimensional problems, the sparse grid technique is often used in place of classical numerical schemes because it is unaffected by the so called curse of dimensionality. The combination technique is then used to further decompose the problem into multiple partial problems that can be solved on many computers independently and in parallel. With an increasing number of parts participating in the computation, the algorithms have to be carefully designed to be resistant against the failure of a few components. This work introduces a way to integrate variable process group sizes into the sparse grid framework SG++, allowing for a more dynamic and thus more error proof combination technique.

In höher dimensionalen Problem wird of the Dünngitter-Methodik anstelle von klassischen numerischen Schematas eingesetzt, weil diese nicht vom so genannten *Fluch der Dimensionen* betroffen ist. Die Kombinations-Technik kann dann zusätzlich verwendet werden, um das Problem in mehrere Teilprobleme aufzuspalten, die voneinander unabhängig auf einer Vielzahl an Computern parallel gelöst werden können. Bei einer steigenden Menge an benutzter Hardware wird es immer wichtiger, den Algorithmus fehlerunanfällig gegenüber Fehler derselben zu gestalten. In diesem Werk werden variable Prozessgruppengrößen als Teil des SG++ Frameworks eingeführt, die es so unter anderem erlauben, eine dynamischere und damit fehlersicherere Kombinationstechnik zu implementieren.

Contents

1 Introduction

When solving physical problems, formulated as partial differential equations, it is almost always imperative to provide accurate results, especially in applications such as plasma & fluid dynamics. The conventional way is to discretize the problem domain into successively finer grids until a satisfactory error term has been reached. A problem, the so called *curse of dimensionality*, is encountered when solving higher dimensional equations.

Motivated by the interest to circumvent this problem, the Sparse Grid approach has been developed for the solution of partial differential equations by Zenger [11] based on an underlying hierarchical basis which has been introduced earlier by Smolyak [10]. It makes use of the tensor product to span a multidimensional, discrete function space. By doing so, in contrast to the conventional, regular nodal basis, the amount of grid points for a mesh of dimension d and norm h is reduced from $O\left(h^{-d}\right)$ to $O\left(h^{-1}log\left(h^{-1}\right)^{d-1}\right)$ while the accuracy of the approximation, under certain smoothness conditions, is only reduced from $O\left(h^2\right)$ for conventional methods to $O\left(h^2log\left(h^{-1}\right)^{d-1}\right)$ with the sparse grid technique. [3] Notice that in the conventional basis the degrees of freedom are exponentially dependent on the dimension d. This is problematic, since doubling the grid resolution exponentially increases the d.o.f., the *curse of dimensionality* is encountered. Sparse grids effectively lift this curse, since the dimension now enters the error term as the exponent of the logarithm of the mesh size instead of the exponent of the mesh size directly.

To further make use of the capabilities of parallel computing on modern hardware, the *combination technique* is employed. This technique decomposes a problem formulated on a sparse grid space into multiple problems on full grids, still with less degrees of freedom than the original full grid problem. By doing this, almost all algorithmic approaches that are valid for the full problem can be reused to solve the decomposed one without the need for specially designed algorithms to fit the sparse grid approach.

The combination technique then dictates how to assemble the sparse grid solution from the solutions obtained on the smaller full grids and from there, the result in the original problem domain can be obtained.

The sparse grid approach is attractive because many of the tools and solvers originally developed for conventional grids can be reused fairly easily. Most optimizations

and techniques applicable for the solution of a conventionally posed problem are effortlessly integrated into the sparse grid approach via the underlying full grid solver.

We will first give an introduction into the theory of sparse grids and the combination technique. Then, the SG++ toolbox, a programming framework applying these technique, will be presented. This work then contributes to SG++ by implementing an inhomogeneous combination technique. Possible communication schemes will be discussed in 3.1 and following. Results in the form of performance tests on the CoolMUC2 linux-cluster at LRZ will be presented in the last section.

2 Background

2.1 Sparse Grids

The standard hat function,

$$\phi(x) := \left\{ \begin{array}{ll} 1 - |x| & \text{if } x \in [-1,1] \\ 0 & \text{otherwise} \end{array} \right\} \tag{2.1}$$

is used to formulate the hierarchical basis.

Consider first only a one dimensional mesh defined on the interval $\Omega = [0,1]$ with a certain mesh size $h_l := 2^{-l}$ for some discretization level l. The so defined grid Ω_l consists of the points $x_{l,i} := i \cdot h_l$ for $0 \leq i \leq 2^l$. To each level and point $x_{l,i}$ is associated an appropriately scaled and shifted version of the standard hat function.

$$\phi_{l,i}(x) := \phi\left(\frac{x - x_{l,i}}{h_l}\right) \tag{2.2}$$

The conventional nodal basis V_l can then be defined as

$$V_l := span\left\{\phi_{l,i} \mid 1 \leq i \leq 2^l - 1\right\} \tag{2.3}$$

where each $x_{l,i}$ corresponds to one basis function which support is the neighboring intervals.

The hierarchical basis W_l shall be defined as

$$W_l := span\left\{\phi_{l,i} \mid 1 \leq i \leq 2^l - 1, i \text{ odd}\right\} \tag{2.4}$$

Figure 2.1: The nodal basis (right) can be represented as a sum of all smaller levels of the hierarchical basis (left)

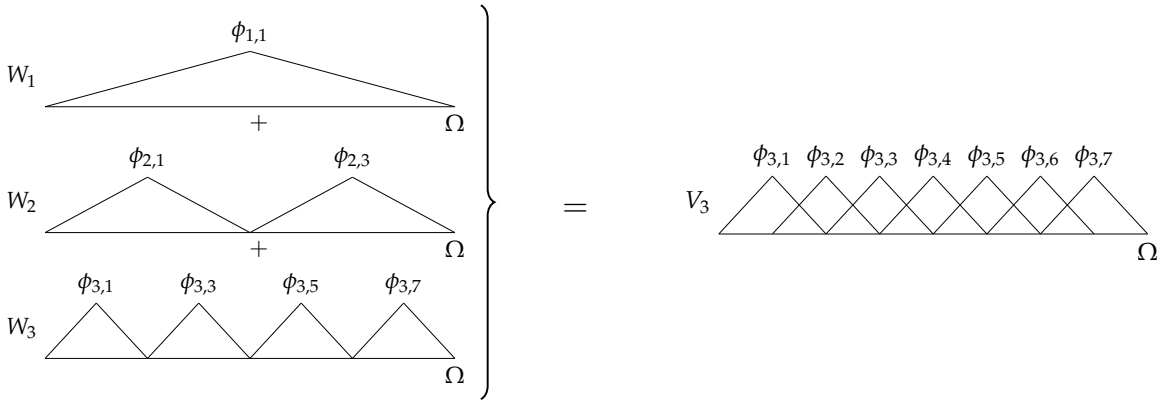
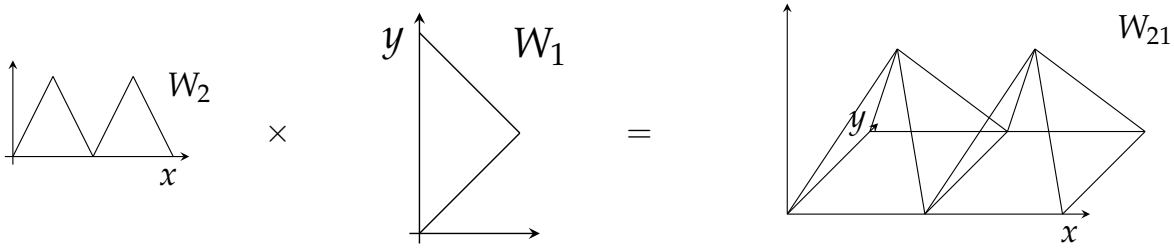


Figure 2.2: The higher dimensional hierarchical basis can be constructed as a product of lower dimensional bases



We notice the relation $V_l = \bigoplus_{k \leq l} W_k$, i.e. each function representable by a conventional nodal basis of level l can alternatively be written as a sum of functions belonging to the hierarchical bases of levels $k \leq l$.

To generalize into arbitrary higher dimensions d, the tensor product can be utilized, where the index is now a d-dimensional vector $\vec{l}$ or $\vec{k}$ and the tensor product is taken component-wise. We write

$$V_{\vec{l}} := \bigotimes_d V_{\vec{l}_d} \tag{2.5}$$

$$W_{\vec{l}} := \bigotimes_d W_{\vec{l}_d} \tag{2.6}$$

One last step remains to the construction of sparse grids. As was seen earlier $V_l = \bigoplus_{k \leq l} W_k$ which generalizes to d dimensions as

$$V_{\vec{l}} = \bigoplus_{\vec{k} \leq \vec{l}} W_{\vec{k}} \tag{2.7}$$

where $\leq$ is understood component wise: $\vec{a} \leq \vec{b}$ iff $\vec{a}_i \leq \vec{b}_i \forall i : 1 \leq i \leq d$. To construct a sparse grid, one choses instead of all $\vec{k} \leq \vec{l}$ only some of them, $\vec{k} \in \mathcal{I} \subseteq \mathbb{N}^d$ resulting in the function space $V_{\mathcal{I}} := \bigoplus_{\vec{k} \in \mathcal{I}} W_{\vec{k}}$.

Let now $f : \mathbb{R}^d \to \mathbb{R}$ be a function defined over the d-dimensional box $\Omega := [0,1]^d$. General rectangular bounds can be accommodated for by rescaling. Choose also an approximation $f_{V_{\mathcal{I}}} \in V_{\mathcal{I}}$.

To overcome the curse of dimensionality, the goal is to reduce the number of degrees of freedom, i.e. grid points of $V_{\mathcal{I}}$, while also constraining the error of approximation $\left\| f - f_{V_{\mathcal{I}}} \right\|$ for some norm $\|\cdot\|$. As has been shown in [2], for the L_2 and L_∞ norm, the optimal choice is $\mathcal{I}^{(1)} := \left\{ \vec{k} \in \mathbb{N}^d \mid \left| \vec{k} \right|_1 \leq n + d - 1 \right\}$ for some n representing a isometric grid of level $n \cdot \vec{1}$. This results in the conventional sparse grid space $V_n^{(1)} := V_{\mathcal{I}^{(1)}}$.

Comparing $V_n^\infty := V_{n \cdot \vec{1}}$, the full grid, with $V_n^{(1)}$, the sparse grid, proofs the claims from the introduction. The full grid is affected by the curse of dimensionality by having $O\left(2^{nd}\right)$ grid points and an error term of $O\left(2^{-2n}\right)$ whereas the sparse grid exhibits only $O\left(2^n n^{d-1}\right)$ degrees of freedom while keeping the error with respect to the L_2 and L_∞ norm smaller than $O\left(2^{-2n} n^{d-1}\right)$.

Still, the O-terms depend on the dimension. Further improvement can be made by optimizing $\mathcal{I}$ for the energy norm $\|\cdot\|_E$ yielding $\mathcal{I}^{(E)}$. The so constructed space $V_n^{(E)} := V_{\mathcal{I}^{(E)}}$ is a subspace of $V_n^{(1)}$, has $O\left(2^n\right)$ degrees of freedom and an error w.r.t the energy norm of $O\left(2^{-n}\right)$. This finally overcomes the issue with dimensionality since neither the complexity nor the error term depend asymptotically on the dimensionality of the problem. The dependence on d is hidden in constants. For the exact construction of $V_n^{(E)}$ the reader is referred to [2].

In practice, implementing a sparse grid solver alone is not viable. Whereas on regular grids differential operators can be converted to finite difference operators with relative ease, this breaks down on sparse grids due to their hierarchical structure. It would be beneficial, if our calculations would still be done a homogeneous grid. Thus, the combination technique was developed to simplify implementation.

2.2 The Combination Technique

The combination technique [4] provides an approximate function on the sparse grid as a linear combination of multiple functions on smaller, independent full grids. This has two direct benefits. Firstly, the computation on the full grids is readily parallelizable. Secondly, conventional solvers, working only on full grids, can be

employed for the partial solutions which need only be combined in the end to yield a full approximation. As a drawback, the total number of degrees of freedom is slightly increased. Additionally, care has to be taken as to the convergence of the solution. Generally, the combined result from the combination technique will not be fully agree with the solution obtained directly from the sparse grid formulation. In practice this is not a problem, since convergence has been proven for the relevant applications.

We write abstractly that a sparse grid approximation $f_{\vec{n}}^{(c)} \approx f$ can be recovered with the formula

$$f_{\vec{n}}^{(c)}(x) = \sum_{\vec{l} \in \mathcal{J}} c_{\vec{l}} f_{\vec{l}} \tag{2.8}$$

for some level set $\mathcal{J}$, $f_{\vec{l}}$ are approximations on the corresponding full grids. The classical combination technique as found in [1] can be written as

$$f_{\vec{n}}^{(c)}(x) = \sum_{q=0}^{d-1} (-1)^q \binom{d-1}{q} \sum_{|\vec{l}|_1 = n-q} f_{\vec{l}} \tag{2.9}$$

It can be straightforwardly shown, by using 2.7 and a few telescopic sum, that, for a given function f, and if $f_{\vec{l}}$ are the suitable full grid approximations, then $f_{\vec{n}}^{(c)}$ is equivalent to the sparse grid approximation. [3]

One must be careful though, since this equivalence does not generally hold for the combination of independantly evaluated solutions of an arbitrary differential equation. Here, $f_{\vec{n}}^{(c)}$ might differ from the solution obtained on the sparse grid. Nevertheless, it has been shown the the same error in convergence can be obtained, as long as the so called *error splitting assumption* is fulfilled by the differential equation. [4]

If we now analyze again the degrees of freedom in the new system, we find that it must be higher than for the sparse grid, since some subspace occur multiple times in the full grids. Indeed, we find $O(dn)$ full grids with $O(2^n)$ d.o.f. individually. This is slightly worse, but still a lot better than $O(2^{nd})$ d.o.f. required for the full grid solution. [6]

Equipped with the theory, we now turn to look on an implementation of this technique, the SG++ framework.

2.3 The SG++ framework

SG++ [8] is a programming framework which provides data structures for solving high dimensional partial differential equation on sparse grids, using the combination technique to distribute and parallelize the workload on a massively parallel system. No underlying full-grid solver is directly included in the library, it is instead abstracted away to be supplied by the user, based on the problem. As a result the framework can be applied independently of the differential equation in question, barring considerations of convergence. A detailed description can be found in [9] and [5], important terms and ideas required to motivate this paper are reproduced in the following.

2.3.1 Process and data layout

Listing 2.1: The essential part of the task interface as found in [5]

```
class Task {
public:
   virtual void run(CommunicatorType lcomm) = 0;
   virtual void init(CommunicatorType lcomm) = 0;
   virtual DistributedFullGrid<CombiDataType>& getDistributedFullGrid()
      = 0;
};
```

The solution of a given problem on a sparse grid is constructed from the result on multiple full grids. Any computation on such a full grid is abstractly captured in a `Task` (see 2.1). Each task represents a function on such a grid, evolving in time. An implementation must provide means to initialize and resume the computation - i.e. advancing time -, and provide access to an underlying distributed full grid. Ideally, the same grid is used for computation and access though this is not always possible. Depending on the solver used, data has to be copied from an internal representation into the `DistributedFullGrid` and back, or it might even be written to and read from a file on disk if no direct access to the data can be offered. The so introduced potential overhead is compensated by the flexibility of the Task interface.

The computational resources are represented as MPI processes. The available processes are grouped in two nested parallelization levels, naturally fitting the combination technique. On a coarse level, each of the full grids can be computed in parallel to the others and is thus associated with one group of processes. On a second finer level, each process in one such group computes only a part of the whole grid.

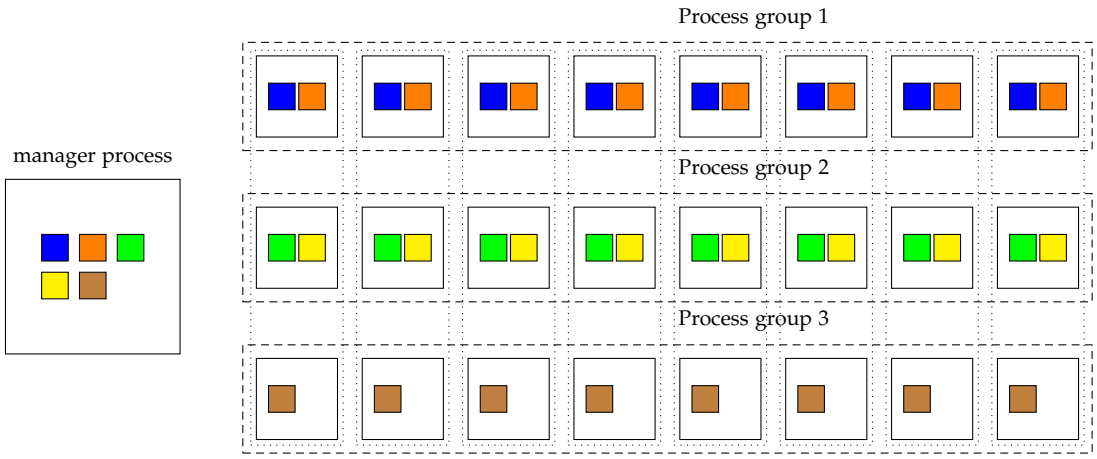


Figure 2.3: Processor layout in SG++ showing the manager-worker pattern for three groups of size 8. The process manager distributes tasks to the process groups. Each process in the group is responsible for a subdomain of the task's grid. The dashed and dotted boxes show the local and global reduce communicators respectively. Adapted from [5]

The domain is geometrically split into subdomains where each process computes only grid points inside one subdomain. We require geometric split to be the same on all groups. Such a decomposition is called *distributed component grid*. Next, an overview over the typical program flow is given.

2.3.2 Program flow with SG++

In each group one process is selected as the *master process*, responsible for communication with a separate, globally unique manager process. This manager's responsibility is to coordinate the computation by sending appropriate signals to the master processes which further broadcast the signal locally in their group over a *local communicator*. After signal-handling is complete, a `READY` signal is propagated in reverse order. The following signals are available:

`RUN_FIRST`	A task is sent alongside the signal and assigned to the process group. The initialization routine `Task::init` of the task is run and the distributed component grid is created and filled with the initial values.
`RUN_NEXT`	Each assigned task is sequentially resumed via `Task::run`.
`COMBINE`	The component grid are globally combined. This is a global action, i.e. all processes cooperate by exchanging data between groups.
`EVAL`	The global solution according to the combination technique is computed from all component grids.

The first signal sent is always `RUN_FIRST`, once for each task that is to be executed on the receiving processes. It is followed by several `RUN_NEXT` signals for each timestep, interleaved with a `COMBINE` signal once every few timesteps. Once the desired time frame has been computed, the solution is extracted from the grid by sending the `EVAL` signal. Since the rest of this work focuses mostly on the `COMBINE` step, this should be inspected in more detail.

2.3.3 The combination step

During the combination step the solution from the different full grids is combined into the sparse grid solution according to the used combination technique. The result is then projected back onto the different full grid spaces and used subsequently as a initial value for the next timesteps.

First, the full grid solution is, in a process called *hierarchization*, projected onto the hierarchical basis. Note that, because the domain geometrically decomposed, this is a group local operation. The contributions from all tasks of the group are added up with the weight according to the chosen combination technique.

This results in multiple distributed sparse grids for each group. The decomposition of these grids onto the processes can generally be chosen freely. One possible choice could be to assign each sparse grid level to one process from the group that stores the whole level. In this paper we will focus on a different, perhaps more natural choice: The sparse grid space is geometrically decomposed exactly in the same way that the full grid space was. Recall that each nodal basis function corresponds to exactly one hierarchical basis function. If a processes did store the nodal grid component, it will also store the hierarchical one.

With this setup, we proceed to combine the solution from the groups. The coefficients of the hierarchical basis have to be simply added up. To implement this efficiently, notice that as a result of the geometric split data must only be exchanged

between processes with equivalent subdomains. Exploiting this, these equivalent processes form so called *global reduce* MPI communicators. Summing the coefficients is then done with copying the data into a buffer and calling `MPI_Allreduce(..., op = MPI_SUM, ...)` on it.

In a third and last step, the hierarchical basis is *dehierarchized* and projected back onto the nodal basis.

Given this framework we now want to introduce inhomogeneous groups. This will violate one assumption made in the second step of the combination step, in particular the subdomains of different groups will not correspond one to one.

3 Implementation of an inhomogeneous massively parallel combination technique

The current state of SG++ requires all process groups to be of the same size. This is not ideal for two reasons. Firstly, different group sizes potentially allow for a better task distribution. Computing tasks with larger grids on larger groups can reduce the average runtime of a single task for one process. These smaller chunks can be used to balance the execution time more between process groups.

The second advantage of allowing different-sized process groups is an increased fault-tolerance. If only one component fails in a system with equally sized groups, the whole group of that component can not be used either until the component is replaced or all groups are sized down to fit a new decomposition. The first option risks not using a substantial amount of remaining available computing resources while the second option can only be realized by globally adopting a new decomposition. All groups must be split up and their processes regrouped.

With nonuniform process group sizes, the fault of one processing unit can be more easily handled by splitting only its group into multiple smaller-sized groups. The decomposition must only be changed locally in one group. Care must be taken, as group sizes below a certain limit can lead to bottlenecks during the combination step (see 3.1).

The rest of this work describes how variable group sizes are realized as an addition to the SG++ framework.

[6]

3.1 Data structure and communication

The biggest change when allowing differently sized process groups must be made in handling the `COMBINE` signal. The difficulty arises during the global reduction. Since

processes in groups of different sizes can't be associated one-to-one, grouping processes according to the geometrical decomposition ceases to work. Were we to allow arbitrary group sizes subdomains of processes of one group could partially overlap subdomains of processes from another. This would complicate global reduction, since some processes would need to send and receive parts of their data with potentially multiple processes from other groups.

To simplify the situation, we will introduce the notion of *compatible* grid decompositions. Let A, B be two process groups containing the processes $a_i \in A, i \in [0, n_a[$ and $b_j \in B, j \in [0, n_b[$. Then the grid decompositions on A and B will be called compatible and, w.l.o.g. we will say that A refines B and B is a generalization of A iff for each process b_j there exists a set $G_j \subseteq A$ of processes in A such that the geometric domains associated with the processes in G_j make up the domain of b_j, i.e. $\Omega(b_j) = \bigcup_{a \in G_j} \Omega(a)$.

We will call the sets G_j *teams* and select one of the processes in this set to be the *team leader*. An MPI communicator is formed for each team.

By limiting the discussion to compatible component grids, it is clear that every group contains the same amount of pairwise distinct teams though the teams are of possible different sizes. Indeed, all smallest groups contain only teams of size 1 while bigger groups must have teams of appropriately larger sizes.

Multiple variants of a global reduce algorithm shall be considered. All of them will exploit the compatibility condition, either of the sparse grid space or the full grid space. We will consider in the following three possible implementations, and compare their benefits and drawbacks.

3.2 Variant 1: Early team gather global reduce

Possibly the most obvious choice to implement the combination step with inhomogeneous groups is to simply join the task grids of each team, formed according to compatible full grid decompositions, on the *team leader* process before combining. Abusing the fact that all processes in a group execute the same tasks, except for being responsible for different subdomains, each full grid found on one of them, must also be present on any other process from the same team.

On each team, when combine is called, the team leader provides a buffer for each task to store the combined distributed component grid from all team processes. Then, the normal combination is executed, except that only the team leaders are participating, using the temporary distributed full grids.

Note that technically, the decompositions of the task grids must not even be compatible for this variant, although it is helpful imagining them as such. We could

pick the same number of processes from all groups and combine the grids of the whole team on them. With some additional care during hierarchization, we still end up with the same hierarchical subspace on equivalent processes in each group. If, for example, we chose to decompose the sparse grid into the hierarchical subspace, as described as Variant 1 in the Local reduction step in [5], we will simply only assign subspaces to the team leaders.

In practice, this approach is unwieldy. Consider that the hierarchization step is executed only on the team leaders while the rest of the processes are idle. We thus propose Variant 2 as an improvement.

3.3 Variant 2: Team gather global reduce

Let us investigate how we could do better than merging the task grids before the global reduction is actually performed. Notice that in case of an variant 1, only the team leaders are involved in the hierarchization, global reduction and dehierarchization while the rest of the processes are waiting. An improvement can be made if, instead of combining full grids, we first hierarchize in parallel the component grids on all group processes simultaneously. Only afterwards are the sparse grids gathered on the team leader processes, which is responsible for all global communication with other groups. Then the flow is reversed: the sparse grids are scattered throughout the team, dehierarchized and projected back on the task grids.

We will still preserve the idea of team leaders - only one process from each group takes part in the global reduction. As such, the distributed sparse grid must be gathered on the team leaders. The amount of communication depends on the chosen decomposition. We shall consider a process group with m processes, a homogeneous team size of g and N grid points. Then the amount of team leader processes is given by $n := m/g$. In the following we analyze the expected performance in parallel to [5].

Assigning the hierarchical subspaces as a whole to processes, each team leader will store on average of $\frac{N}{n}$ grid points which he receives from $m-1$ different processes. Each non-team leader process sends $\frac{N}{mn}$ grid points to the team leaders. The time to send a message will be modeled as $t = L + K/B$, given a latency L, a datasize of K and a bandwidth of B. Then the total time to received all messages on the team leader processes is given by

$$t_{total} = (m-1) \cdot (L + \frac{N}{mnB}) \sim mL + \frac{N}{Bn} \tag{3.1}$$

Unsurprisingly, the asymptotic behavior is the same as if the grid was distributed over the whole group. Significantly though, for smaller m the runtime is impacted

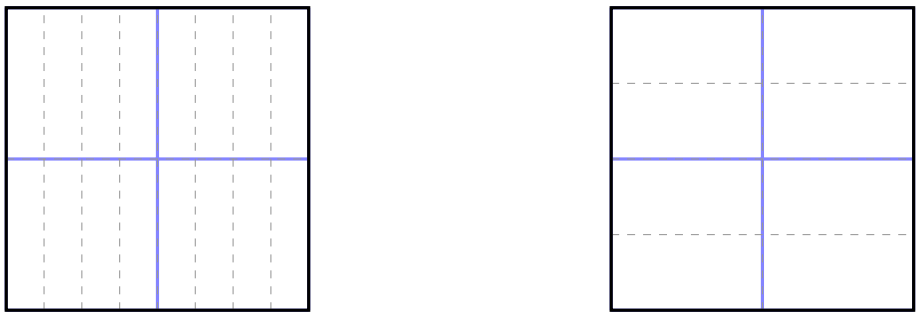


Figure 3.1: The dashed lines show two geometrically decomposed grids. On the left, the grid is anisometrically decomposed in 16 subdomains. On the right, the grid is split into only 8 subdomains. The two decompositions are not readily compatible to each other, but we can find a decomposition into 4 parts that generalizes both of them (marked by the blue lines).

more heavily bv the amount of team leaders $n < m$ and the term $\frac{N}{Bn}$. This makes this approach even less favorable. We turn towards a different sparse grid decomposition.

If instead, we decompose the sparse grid geometrically equivalent to the full grid, data must only be exchange locally inside each team. Modeling with the same tools as before we find that each process sends $\frac{N}{m}$ grid points to the team leader, who receives data from $g-1$ team members, leading to a predicted runtime of

$$t_{total} = (g-1) \cdot (L + \frac{N}{mB}) \sim gL + \frac{N}{Bn} \tag{3.2}$$

The interesting fact to notice here is that the runtime actually benefits from larger group sizes, as long as the team size is kept fairly low (see also 4). An implementation of this behavior has been added to the SG++ library as a possible local reduce method to call after hierarchization. Keep in mind that we must require that all groups have the same sparse grid decomposition. As the resulting domain on the team leader is simply the union of the domain of all team processes, the original full grid decomposition of each group must be *compatible* with the sparse grid decomposition.

It should be emphasized that still, the decompositions of the full grid, relevant for the underlying full-grid-solver, can vary from group to group. This can be abused. One can imagine to have larger groups with anisotropic decompositions in one coordinate direction that are computing equally anisotropic task grids such that each process individually works on a relatively isotropic grid. Having such a "specialized" group for each coordinate, during the combination step the sparse grid is decomposed isotropically, compatible with each individual group decomposition. This results in small team sizes (only the anisotropic part of the group parallelization).

Although not part of the implementation of this work, it is not required that the resulting sparse grid decomposition on the team leaders is one of the decompositions of an actual group. In that case, all teams on all groups would consist of more than one process. This is shown in 3.1.

Designing such a load balancing is out of scope of this work, as that would require a sophisticated runtime model, specialized with an understanding for the underlying solver, equipped with a prediction based on both the (an)isotropy of the grid and the group parallelization. It should be noted that this has already been explored partially, but not investigating the effect of different group decomposition for the GENE software in [5].

3.4 Variant 3: Joint global reduce

As team sizes group, variant 2 will exhibit the same linear asymptotic slowdown as variant 1. It could be beneficial if all processes would take part in the global reduction, without locally gather data first. To simplify the implementation of an `Allreduce` operation with different distributed sparse grids on each groups, we assume that the decompositions are pairwise compatible. This is a stronger requirement than in variant 2 where we only assumed one decomposition generalizing all others. The idea is that all communication is between groups, but only between associated teams. More importantly, a communication structure is now tree like.

At the root level of the tree we find the coarsest decomposition. Each subdomain is represented by a node. The pairwise compatibility induces a total ordering on the decompositions, where $A < B$ if A refines B. A node in the tree is then connected to the set of nodes that correspond to its domain in the refinement. Such a tree is visualized in 3.3.

How would communication look? The MPI `Allreduce` operation most commonly employs a binary tree structure (given that addition is an associative operation). Data is propagated up the tree, gathered at the root and propagated back down the opposite tree side. We propose a similar structure, with an additional operation to propagate data between levels of decompositions.

First, all processes on the same decomposition level form a binary tree and data is propagated to the root of that tree. Then, the root nodes of the tree reduce the data among them. The tree structure of the decomposition is used as the communication tree. Data is propagated up the tree by gathering the data from the lower level and then reducing with the node's own data. Propagating down the tree is then done by scattering the data according to the refined decomposition. In the last step, data is sent down the binary tree of the individual levels.

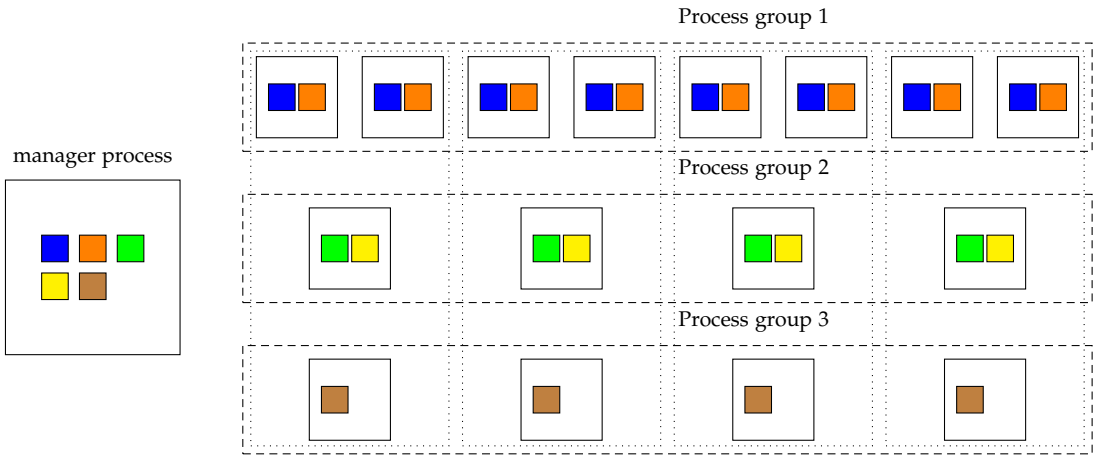


Figure 3.2: Processor layout for joint global reduce. Group 1 has a size of 8, group 2 and 3 consist of 4 processes each. Compared with homogeneous group sizes, the global reduce communicators now include more than one process from group 1.

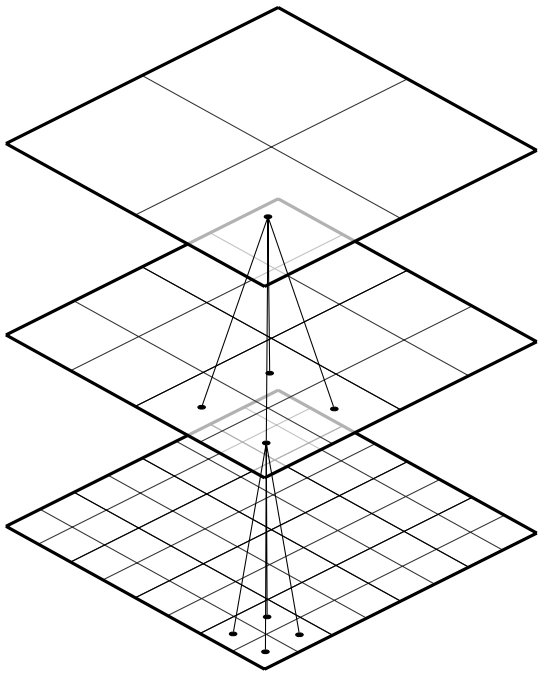


Figure 3.3: A sketch of the tree like structure of three pairwise compatible geometrical grid decompositions. The first, coarsest level has 4 subdomains, the second 16, the third has 64, corresponding to equally sized groups. Only some branches are shown.

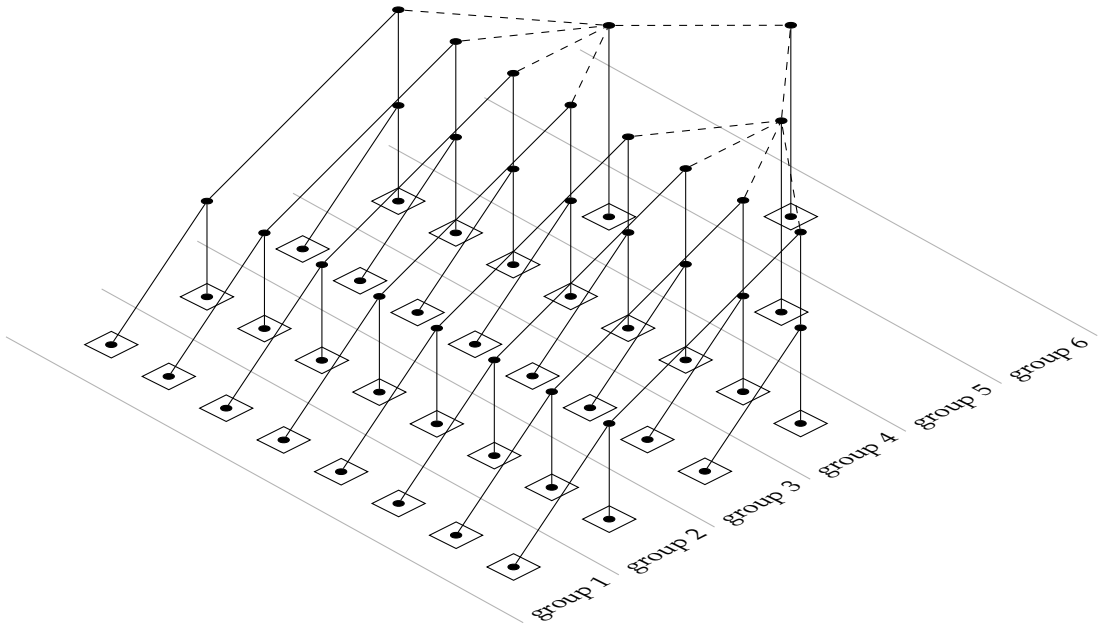


Figure 3.4: Possible all reduce communication structure. Dashed lines are used to mark communication between different decomposition levels, full lines are used where a normal Allreduce binary tree is used.

The main drawback of this variant is that pairwise compatibility is quite a heavy restriction on the possible decompositions. For example, it is now *not* possible for one group to be decomposed mainly in one coordinate direction while another is decomposed mainly in a different one, as is the case for anisotropic decompositions in multiple directions. The reason for this is that instead of the all reduce operation being executed on all coarsest, most general decompositions in parallel, it must now be on a finest decomposition that refines all group decompositions.

Another drawback is the complexity of the implementation. Realizing the tree-like communication structure necessary to implement the required `Allreduce` operation is complicated. This also comes with the loss of optimizations employed by MPI internally in its built-in `Allreduce` operation.

As can be seen in 3.4, the bottleneck in communication is certainly the bandwidth of the level tree, most importantly in the higher levels because data is appended, not reduced, when being sent up the tree. It might be possible to anticipate and counter this issue by employing a fat tree communication structure [7]. No implementation or further discussion of this variant is part of this work, as reasonable assumptions can be made that justify small team sizes in the present situation. We shall now turn our attention to the results of implementing variant 2.

4 Results and Conclusion

A simulation has been set up to test the influence of different group sizes on the combination step. The simulation has been run on the LRZ Linux cluster system. Since we are only interested in the timings for the combination step, all other other computations would be a waste of resources and have been disabled. The tested algorithm is variant 2 3.1 of the global team reduce operation.

The simulation was run using a 5 dimensional, fixed sparse grid decomposition and varying team sizes. The following configuration was tested, giving 8 simulations runs. In each run the combination step was run 1000 times to account for jitters and inconsistencies caused by the host system. 513 processes on 19 nodes of the mpp2 cluster were reserved to run the program.

#	$\vec{l}_{min}$	$\vec{l}_{max}$	subspace count
1	$(3,1,4,4,4)$	$(3,1,6,6,6)$	9
2	$(3,1,5,5,5)$	$(3,1,7,7,7)$	9

Table 4.1: Combination techniques used in the testing

#	process grouping	team size	group parallelization	total # of processes
1	$256+256$	1	$(1,1,8,8,4)$, $(1,1,8,4,4)$	513
2	$256+2\times128$	2	$(1,1,8,8,4)$, $(1,1,4,4,4)$	513
3	$256+4\times64$	4	$(1,1,8,8,4)$, $(1,1,4,4,2)$	513
4	$256+8\times32$	8	$(1,1,8,8,4)$, $(1,1,4,2,2)$	513

Table 4.2: Group constellation used in the testing. Group parallelization refers to the geometric subspace decomposition used. The processes in each group are responsible for one cell in the regular grid with the given number of cells in each dimension.

The results from the testruns can be seen in 4.1. We can use a team size of 1 as our baseline figure. This corresponds to the case of homogeneous group sizes, i.e. when no teams are needed and the groups are compatible out of the box.

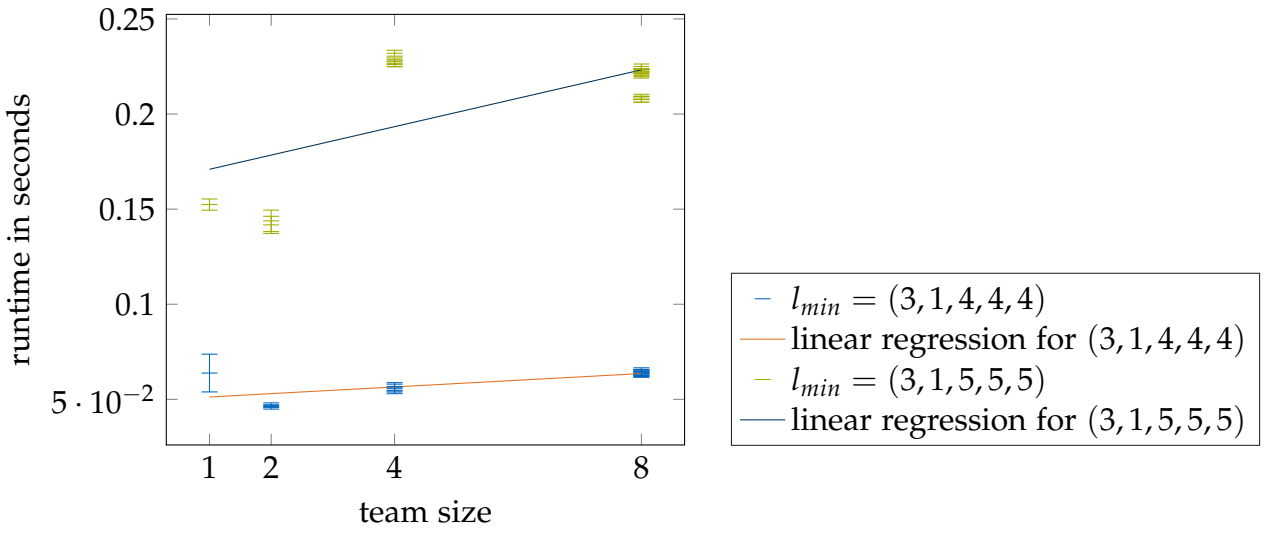


Figure 4.1: Runtime results timing the combination step. Each process group corresponds to one data point, hence for bigger team sizes more data points are present.

With a growing number of process groups, the data present on each group is reduced accordingly. Take note that the task grids needed for the used combination technique are distributed to the teams in a round robin fashion.

We first have to address two problems in our dataset. Firstly, the big standard error and deviation from the linear regression for $l_{min} = (3,1,4,4,4)$ is most likely caused by the relatively small choice of l_{min}. The group process size is almost as big as the grid has degrees of freedom, causing relatively erratic and unreliable behavior.

The second irregularity can be seen in the data for a team size of 8, looking at $l_{min} = (3,1,5,5,5)$. Here, the data points separate into piles, one with a bit smaller runtime than the other. This can probably be attributed to the round-robin assignment of task grids leading to different grid sizes between the process groups from each pile.

Nevertheless, we see limited growth in runtime when increasing the team size, at least for modest team sizes. This allows us to conclude that choosing group decompositions that lead to small team sizes is a viable option without detrimental drawbacks in the combination step. One could, for example, use this to implement a fault-tolerance scheme. Imagine one process of a big group failing. Instead of not being able to use all processes in the group due to requiring exactly matching group sizes, one can now pick some constant maximum team size, say 16, and split the faulty group into up to this many smaller groups. Only one of those smaller groups contains the faulty process and can not be used, the rest can be recovered.

We conclude that inhomogeneous group sizes, at least to some limit, are a viable strategy that can be employed for better fault tolerance, or could even be used for anisotropic group decompositions.

5 Future work

Several opportunities exist for future works. Possibly the most natural would be to further explore Variant 3 of the team global reduce operation. Intuitively, it is expected that this variant scales sublinearly with the group sizes, because communication is done on a tree. Further analysis of the message scheme is required to support this claim.

Another avenue is to continue with variant 2 of the implementation and simulating bigger team sizes. Although theoretically showing the same asymptotic behavior as variant 1, team sizes being substituted for group sizes, practical limits might lie beyond what has been demonstrated in this paper.

At last, it might be explored if and how anisotropic group decompositions could help speed up computation on anisotropic grids.

List of Figures

List of Tables

Bibliography

[1] H.-J. Bungartz, M. Griebel, D. Röschke, and C. Zenger. "Pointwise Convergence Of The Combination Technique For Laplace's Equation." In: *East-West J. Numer. Math* 2 (1994), pp. 21–45.

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

[3] J. Garcke. "Sparse Grids in a Nutshell." In: *Sparse Grids and Applications*. Ed. by J. Garcke and M. Griebel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 57–80. ISBN: 978-3-642-31703-3. DOI: 10.1007/978-3-642-31703-3_3.

[4] M. Griebel, M. Schneider, and C. Zenger. *A Combination Technique For The Solution Of Sparse Grid Problems*. 1992.

[5] M. Heene. "A massively parallel combination technique for the solution of high-dimensional PDEs." PhD thesis. University of Stuttgart, 2017.

[6] M. Heene and 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, and W. E. Nagel. Cham: Springer International Publishing, 2016, pp. 165–186. ISBN: 978-3-319-40528-5. DOI: 10.1007/978-3-319-40528-5_8.

[7] C. E. Leiserson. "Fat-trees: Universal Networks for Hardware-efficient Supercomputing." In: *IEEE Trans. Comput.* 34.10 (Oct. 1985), pp. 892–901. ISSN: 0018-9340.

[8] D. Pflüger. *SG++: General Sparse Grid Toolbox*. http://sgpp.sparsegrids.org/. 2008–2016.

[9] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. München: Verlag Dr. Hut, Aug. 2010. ISBN: 9783868535556.

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

[11] C. Zenger. "Sparse Grids." In: *Parallel Algorithms for Partial Differential Equations, Proceedings of the 6th GAMM-Seminar Kiel 1990*. Friedrich Vieweg & Sohn Verlagsgesellschaft mbH, 1991, pp. 241–251. ISBN: 978-3528076313.