Communication Reduced Parallel Multigrid: Analysis and Experiments

Marcus Mohr  Ulrich Rüde *

October 12, 1998

Abstract

The most common technique for the parallelization of multigrid methods is grid partitioning. For such methods Brandt and Diskin have suggested the use of a variant of segmental refinement in order to reduce the amount of inter-processor communication. A parallel multigrid method with this technique avoids all communication on the finest grid levels. This article will examine some features of this class of algorithms as compared to standard parallel multigrid methods. In particular, the communication pattern will be analysed in detail.

1 Introduction

Many tasks in science and engineering require the solution of partial differential equations (PDE). The numerical treatment of these, namely the discretisation with finite differences or finite elements, leads to the problem of solving large systems of linear equations with sparse matrices. Multigrid methods are known as fast and efficient iterative solvers for such problems.

The progress in the treatment of PDEs is confronted with a continuous growth in size of the respective tasks. Three-dimensional problems may easily involve several millions of unknowns and in a non-stationary and/or non-linear case these systems must be solved thousands of times. To solve such large problems, a parallelization is necessary to split the work between different processors.

There is a great variety of different parallel architectures, ranging from clusters of workstations to massively parallel machines. A categorization is given in [13]. In this paper we will consider the case that the number of processors is significantly smaller than the number of grid points.

With this background the most common approach to the parallelization of numerical algorithms is grid partitioning. In the special case of multigrid methods nested levels of grids are employed.

This approach introduces the need to communicate values related to the points on or near the inner boundaries between the processors. The cost of communication naturally limits the possible speedup of a parallel method. This cost can be alleviated by sophisticated programming techniques e.g. by overlapping communication with calculation.

*Institut für Mathematik, Universität Augsburg, Universitätsstrasse 14, D-86139 Augsburg
Generally the communication cost to transfer, in one message, $W$ data values (words) between two processors is modelled by a two-parameter function,

\[ T = \alpha + \beta W. \]

Here $\alpha$ is the start-up time for the communication, while $\beta$ represents the time necessary to transfer one word. Both parameters are assumed to be independent of the message length. Employing this model, the overall time spent with communication is on the one hand determined by the architectural parameters $\alpha$ and $\beta$, and on the other hand by the total number of words that must be transmitted and the number of messages that occur.

In multigrid the number of messages is proportional to the number of subgrids and therefore independent of the grid level the communication occurs on\(^1\). The number of words on the other hand is strongly related to the coarseness of the respective level since it is coupled to the number of interface points.

This leads to another aspect, namely the parallel efficiency of the algorithm, determined by the ratio between communication and calculation. This ratio is directly proportional to the ratio of volume and surface of the subgrids and therefore becomes worse on the coarser grid levels. This led to the development of several techniques, like for instance coarse grid agglomeration, multiple coarse grid and concurrent algorithms, see e.g. [10], and methods that employ different cycle schemes like e.g. the U-cycle [15].

As long as one demands that the results of the parallel algorithm must be identical with that of its sequential pendant, a substantial reduction of the communication cost cannot be achieved. Only releasing this demand makes a reduction possible, as it allows for instance not to adjust all subgrids after every relaxation step (inexact block solvers). In [9] this has been shown to work well for some simple problems. Nevertheless one has to be careful with such changes as not to destroy the good convergence properties of multigrid.

A radical approach to reducing fine grid communication has been suggested by Brandt and Diskin in [4]. Here an algorithm that completely eliminates the need for inter-processor communication on several of the finest grids is presented. This is achieved by the use of segmental refinement type procedures similar to those in [2].

In this paper we will examine a variant of the algorithm described by Brandt and Diskin. We want to depict some of its characteristics and problems and compare its communication requirements to that of a “conventional” parallel multigrid algorithm.

The rest of this paper is organized as follows. In section 2 we will compare two different approaches to the partitioning of discretization grids for iterative solvers, especially multigrid methods. In section 3 we will describe the basic structure of the algorithm by Brandt and Diskin and examine the possible time saving of this class of methods.

The communication pattern will be analysed in detail in section 4 by means of the two level version of the algorithm and section 5 will give some numerical results. Finally section 6 will end this paper with some conclusions.

2 Two Approaches to Grid Partitioning

A natural approach to the parallelization of grid applications is grid partitioning. Here the problem grid is divided into several smaller subgrids which are assigned to the different processors.

In the following we want to compare two different concepts for grid partitioning with respect to the model problem below. Both approaches allow for the parallel algorithm to

\(^1\)This of course disregards coarse grid agglomeration techniques
have at any stage the same solution values as its sequential counterpart. Our main interest in this will be the total amount of communication. The differences between the two approaches lie in the way the grid is decomposed and the way that data are stored.

2.1 Model Problem

We consider a multigrid algorithm for the solution of elliptic boundary value problems in 2D. The finest grid shall consist of $n_x \times n_y$ points, boundary points included, where $n_x - 1$ is a power of 2. We have a logical process grid with $p_x \times p_y$ processors where $p_x, p_y \geq 1$. We will also assume that $p_x$ and $p_y$ are powers of 2.

The elliptic problem is discretized with a five- or nine-point-stencil, and we use full-weighting for restriction and bilinear interpolation as prolongation operator. For the solution we use multigrid in the form of coarse grid correction (CGC).

2.2 Overlapping Domain Decomposition

One of the most commonly used and easiest to understand concepts to overcome the difficulties of interfaces in the grid partitioning approach to parallel multigrid is the use of so-called ghost nodes.

For this approach we divide our rectangular grid into as many subgrids as processors we want to use. We choose the partition in a way that every grid point belongs to only one subgrid. This approach is also known as node oriented decomposition in finite element methods. Furthermore the subgrids shall again be rectangular and nearly equal in size. While the former aspect eases implementation and minimizes the length of the interfaces the latter is important for load balancing.

Each processor is then assigned one subgrid. For our problem every operation (e.g., a relaxation step) on a grid point requires only function values from the grid points itself and from its direct neighbours. This means that for points near the interfaces, function values from neighbouring subgrids/processors are required.

To handle these dependencies we place an overlap area around each subgrid. We call these new points ghost nodes and use them to mirror the solution values of the corresponding points in the neighbouring subgrids. If every ghost node is updated before its function value is used for a calculation we will obtain the same algorithmic results as on a single processor.

Note that the thickness of the overlap area depends on the difference-star and the restriction operator that is applied. For our model problem a thickness of one grid point is sufficient.

For multigrid algorithms, all grid levels are decomposed in the same fashion. This is done in a way, that the subdomains on the different levels form a nested hierarchy.
On each grid level we can distinguish four algorithmic steps that may require communication. These are smoothing, defect calculation, the fine-to-coarse transfer and the coarse-to-fine transfer.

Before we look at these steps in detail, let us consider the amount of communication needed to update the solution values at the ghost nodes. Since the new value for a ghost node is completely known to its corresponding grid point, the amount of data values to be transmitted equals the number of ghost nodes in the partition. This is

\[ 2 \left( (n_y - 2) + (p_y - 1) \right) (p_x - 1) + 2 \left( (n_x - 2) + (p_x - 1) \right) (p_y - 1) \]

The next question is the number of messages needed for the update step. Consider a single processor. The new solution values for its ghost nodes are stored on eight neighbouring subgrids, four orthogonal neighbours and four diagonal neighbours. For two reasons a direct communication with the diagonal neighbours is not very attractive. On the one hand the network topology may favour communication along the grid lines. More importantly the diagonal messages would contain only one data value, and as can even be seen from our simple transfer time model (1), communications with few words are very inefficient on most machines.

Fortunately such diagonal communication can be avoided using the transfer model shown in figure 2. Here the fact is exploited, that a corner ghost node of one subgrid is also a non-corner ghost node in a straight-line neighbour subgrid. By applying this transfer model we have the same amount of data values to transmit but approximately half the number of messages. Which is then

\[ 2 \left( (p_x - 1) p_y + p_x (p_y - 1) \right) . \]

With this preparation let us now study the individual steps of the algorithm possibly requiring communication.

Relaxation

Assume that some initial guess is given and the ghost nodes are updated. Then with some smoothers (like e.g. weighted Jacobi-method) each processor may perform independently one relaxation sweep over its subgrid. After all processors have completed this, the information at the ghost nodes must be updated, either for the next sweep or for the defect calculation.

If we use red-black Gaúß-Seidel for smoothing we have to update the ghost nodes corresponding to red points after the first half sweep, so that each processor knows them for the next sweep, the one over the black points. After that sweep has been done the black ghost nodes must be updated. At this the total number of data values stays the same, but the number of messages doubles.

If we use a five-point-stencil the ghost nodes at the four corners need not be updated after the smoothing step, because these values are not required for relaxation or defect calculation. So the number of data values to transmit is reduced by \( 2(p_x - 1)(p_y - 1) \).

Defect calculation

The pre-smoothing steps are followed in the CGC–scheme by the defect calculation. Since we have updated the (relevant) ghost nodes after each relaxation sweep, we do not have to do this prior to defect calculation and the processors can independently complete this task.
2 TWO APPROACHES TO GRID PARTITIONING

Starting Position

A  ● ● ○ ○ ○ ○ ○ ○ ○ ○
B  ● ● ○ ○ ○ ○ ○ ○ ○ ○
C  ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
B  ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
C  ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
D  ○ 0 0 0 0 0 0 0 0 0

1 2 3 2 3 4

1. West - East - Transfer

A  ● ● ○ ● ○ ○ ● ○ ○ ○ ○
B  ● ● ○ ● ○ ○ ● ○ ○ ○ ○
C  ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
B  ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
C  ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
D  ○ 0 0 0 0 0 0 0 0 0

1 2 3 2 3 4

2. East - West - Transfer

A  ● ● ○ ● ○ ○ ● ○ ○ ○ ○
B  ● ● ○ ● ○ ○ ● ○ ○ ○ ○
C  ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
C  ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
D  ○ 0 0 0 0 0 0 0 0 0

1 2 3 2 3 4

3. North - South - Transfer

A  ● ● ○ ● ○ ○ ● ○ ○ ○ ○
B  ● ● ○ ● ○ ○ ● ○ ○ ○ ○
C  ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
B  ○ ● ○ ● ○ ○ ● ○ ○ ○ ○
C  ○ 0 0 0 0 0 0 0 0 0

1 2 3 2 3 4

4. South - North - Transfer

A  ● ● ○ ● ○ ○ ● ○ ○ ○ ○
B  ● ● ○ ● ○ ○ ● ○ ○ ○ ○
C  ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
B  ○ ● ○ ● ○ ○ ● ○ ○ ○ ○
C  ○ 0 0 0 0 0 0 0 0 0

1 2 3 2 3 4

Figure 2: Transfer model for the overlapping domain decomposition with ghost nodes; shown is the update of ghost nodes at the intersection of four subdomains.
Restriction

We want to use full weighting as restriction operator. Therefore each processor must know at each coarse grid point of its subdomain the defect values for all neighbouring fine grid points. As one can easily see, there are points near the interfaces, where this is not the case. To solve this problem one could simply transfer the missing defect values and store them at the ghost nodes. A more efficient way is to let every processor calculate its share of the restricted defect for such a point and to transfer only those values. This is possible due to the linearity of the restriction operator. Without going into further detail we state that with this ansatz we have to transfer

\[
\frac{n_x - 3}{2} (p_y - 1) + \frac{n_y - 3}{2} (p_x - 1) + 2(p_x - 1)(p_y - 1)
\]

words of data in \((p_x - 1)p_y + (p_y - 1)p_x\) messages.

Prolongation

We are using bilinear interpolation as prolongation operator. If we now assume, that the ghost nodes on the coarse grid have been updated after the last smoothing step on that level, we do not need any further communication to calculate the correction on each subdomain. If the ghost nodes on the fine grid have not been changed since the last smoothing step on the fine level we can even perform the correction step on the ghost nodes without data transfer between the processors.

2.3 Non-Overlapping Domain Decomposition

In the non-overlapping domain decomposition approach (NODD), ghost nodes are not needed, instead a subtle data structure and re-interpretation of the data stored in the interface nodes is used.

In this approach, the grid is partitioned into rectangles of almost equal size as before. Here however, the subdomain boundaries coincide with grid lines, so that the nodes on the boundaries belong to all adjacent subdomains. Speaking in finite element terms, the decomposition is chosen in a way such that every element belongs to a unique subdomain, hence the name element oriented decomposition. For an application of this approach to problems involving finite elements and unstructured grids, see e.g., [6].

In our case of structured grids we get three classes of points. Interior points, being part of one unique subgrid, edge points belonging to two and cross points belonging to four subgrids, as can be seen in figure 3.

![Diagram of non-overlapping domain decomposition](image)

Figure 3: Non-overlapping domain decomposition for the case of an unstructured (left) and a structured grid (right)
For the multilevel algorithm all grids are partitioned in the same fashion and each processor is assigned a nested sequence of subgrids. With the NODD-method, the boundaries of a coarse level subgrid will coincide with the boundaries of its corresponding fine level subgrid.

To handle the local dependencies of operations on or near border points the overlapping domain decomposition (ODD) uses ghost nodes while the NODD employs two different data structures. How does this work? If we have to store a global vector of grid values we will break it up into pieces and each processor will store a local vector with values for the points of its subgrid. But what shall be done with points belonging to more than one subgrid?

One possibility is to store the point’s value on every subgrid it belongs to. In this case the global vector is said to be of overlapping type. Formally the splitting of the global vector $v$ can be expressed by $v_i = M_i v$ with $v_i$ being the part stored on the $i$th processor and $M_i$ being a boolean matrix.

Another way is to store the global vector as adding type. Here the value $v(p)$ of the vector $v$ at a grid point $p$ is split between all the processors the point belongs to. So if e.g. $p$ belongs to processors $A$ and $B$ then $A$ will store a value $v_A(p)$ and $B$ a value $v_B(p)$ which add up to the global value $v(p) = v_A(p) + v_B(p)$. So when $n_p$ is the number of processors, then the global vector $v$ can be restored from the local representations $v_i$ through

$$v = \sum_{i=1}^{n_p} A_i^T v_i.$$  

For our parallel multigrid algorithm the approximate solutions and the corrections will be vectors of overlapping type, while the right hand sides and the defects will be vectors of adding type.2

We will now show, how these partition and data structures allow an elegant parallel implementation of a multigrid algorithm and examine its communication cost.

Relaxation

Let us first consider a smoother of Jacobi-type. Here the new approximate solution value $u^{k+1}(p)$ at a grid point $p$ is determined by $u^{k+1}(p) = u^k(p) + c(p)$. Where the correction $c(p)$ is calculated using the old solution values $u^k$ at the point itself and its direct neighbours as well as the value $f(p)$ of the right hand side.

For interior points the calculation of the correction can be carried out by the appropriate processors alone. The situation is different for edge and cross points. Here neither all required solution values $u^k$ are known to the processor nor is the value $f(p)$ of the right hand side, for $f$ being a vector of adding type. But since we are dealing with linear equations, every processor to whom an interface point belongs can take the values known to it and calculate its part of the correction value for it. Together these parts will sum up to the global correction value for the point.

Thus if each processor determines a local correction vector $c_i$ from the locally available data, the global correction vector $c$ can be determined by $c = \sum_{i=1}^{n_p} A_i^T c_i$. So we can calculate the global correction as a vector of adding type without communication. However, before we can calculate $u^{k+1} = u^k + c$ we have to transform $c$ into an overlapping type vector.

Such a transformation can be performed by exchanging the values of the vector corresponding to the interface points. Applying a separated transport scheme, for the same reasons as in the ODD-case, we need

$$2\left[(p_x - 1)p_y + p_x(p_y - 1)\right]$$

2In FEM the stiffness matrix will be stored in an overlapping fashion.
messages to transmit

\[ 2 \left\{ [(ny - 2) + (p_y - 1)(px - 1) + [(nx - 2) + (p_x - 1)(py - 1)] \right\} \]

data values. Note that this is exactly the same amount as for the update of the ghost nodes in the ODD-case. Although the length of the messages will slightly differ between the two approaches we can from our transfer time model (1) expect to measure nearly the same communication times for the update of ghost nodes as for the transformation of an adding to an overlapping type vector.

As in the ODD-case a Gauß-Seidel-type smoother with red–black-ordering requires further considerations, because calculating \( u^{k+1}(p) \) for a black point requires knowledge of \( u^{k+1} \) for its red neighbours. It could be implemented by first calculating a correction for the red points, transforming that correction to overlapping type, applying it to the red points and then calculating a correction for the black points, transforming and applying it.

Although the number of data values to be exchanged would remain constant the number of messages would double with respect to a Jacobi-type smoother. However the following variant requires only one communication step:

1. Calculate correction for the red points.
2. Calculate correction for the black points omitting the ones being direct neighbours of interface points.
3. Transform correction vector from adding type to overlapping type.
4. Calculate correction for remaining black points.
5. Apply correction to all points.

For the next two steps within our multigrid algorithm we must consider the calculation and restriction of the defect.

**Defect calculation & Restriction**

The defect will be stored as vector of adding type and each processor can calculate its part of it separately. Since the right hand side on the coarse grid will also be of adding type it is not necessary to transform the defect. And for the same reason the restriction of the defect, using the full-weighting operator can be done without communication.

**Prolongation**

After the coarse grid problem has been solved, its solution is available as vector of overlapping type. If bilinear interpolation is used the coarse grid correction on the fine grid can be calculated without communication.

### 2.4 Remarks and Comparison

Comparing the two approaches to grid partitioning, we see two advantages of the non-overlapping domain decomposition. The first one is, that it requires less communication. In both approaches we have to exchange the same amount of data values, distributed almost equally on the same number of messages, after each smoothing step. But in the ODD-ansatz the restriction of the defect requires additional communication.
In our situation NODD requires some less communication, but in general this depends on the interpolation and restriction operators used.

The second advantage lies within the geometry of the decomposition itself. For multigrid methods the grids are commonly chosen to consist of $(2^k + 1) \times (2^k + 1)$ points, where $k$ is an index for the specific grid level. With the ODD optimal load balancing with subgrids of exactly the same size is not possible, since the center grid lines with $n = 2^{k-1} + 1$ can only belong to either one of two partitions.

While this is only a minor nuisance, the fact that the interfaces on the different grid levels do not coincide is a bigger problem, since it complicates the implementation of the restriction and prolongation operators. In the NODD--approach the partitioning can be done in such a way, that a hierarchy of nested subgrids will have the same boundaries on all levels.

3 An Approach to Reducing Communication

3.1 Algorithm of Brandt & Diskin

In this part of the paper we will review an idea of Brandt and Diskin for multigrid methods with a reduced need for data exchange between processors. This idea can be used to construct algorithms which do not communicate between partitions of the finest grid levels. Since the bulk of communication takes place on the fine grids, it may be especially attractive to save this cost. The effectiveness of such a technique depends on specific machine and problem parameters. The potential benefits are most impressive when communication is slow and expensive with respect to calculation.

In [4] Brandt and Diskin introduced a parallel multigrid algorithm completely without interprocessor communication on several of the finest levels. This was achieved by the use of segmental-refinement-type procedures, which were originally proposed to overcome storage problems on sequential computers, cf. e.g. [2]. The idea behind segmental refinement lies in a special view of the multigrid method. The more traditional direction is to interpret the processing on coarser grids as a means to eliminate low-frequency error components for which relaxation on the fine grid would be inefficient.

But if we employ the Full-Approximation-Storage (FAS) scheme, the problem on the $(k-1)$-th level can be written in the form

$$L^{k-1}u^{k-1} = f^{k-1} + \tau^{k-1}_k$$

with

$$\tau^{k-1}_k = L^{k-1}J^{k-1}u^{k-1} - \hat{J}^{k-1}L^k u^k$$

and the processing on the fine grid can be viewed as a correction to the coarse grid. Because of the "defect correction" $\tau^{k-1}_k$, the solution $u^{k-1}$ on the coarse grid will have fine grid accuracy.

Segmental refinement exploits this feature of the "corrector" in the following way. Instead of computing $\tau^{k-1}_k$ globally, the domain is split into segments, where $\tau^{k-1}_k$ is approximated locally. Furthermore, the segments are activated only temporarily, that is, fine grid information like $u^k$ is not stored, except for the segment that is just in processing. Of course this introduces some errors. However, since high frequency components, which are characteristic for the fine grid, do not spread far, the effect should be small.

The advantage is that we need only store values related to one segment of the fine grid at one specific time, so we can escape memory limitations. But the idea is also attractive for parallel implementations. Here every segment of the fine grid can be dealt with by a separate
process and since we neglect the dependencies between the separate fine grid solutions, the need for communication is greatly reduced. We only have to correctly set up the fine grid problems and combine the correction values.

In [4] Brandt and Diskin showed that this idea can be extended to include several of the finest grid levels without a loss of accuracy larger than the discretization error, provided there is sufficient overlap. Their algorithm can be described as follows:

- Starting from a hierarchy of grids, in a preliminary step all levels, except the coarsest one, are decomposed into as many overlapping subdomains as processors are available. This is done in such a way that we get sequences of nested subdomains, where a subdomain on a coarser grid occupies a larger area than the corresponding subdomain on the next finer grid.

- Each such sequence is then assigned to a processor and on every level initial values are chosen.

- Each processor then starts on its sequence of grids a standard V-cycle, descending through its grid hierarchy. In this process it does not exchange data with its neighbours.

- When the second coarsest level is reached, the local (approximate) solutions from all processors are used to formulate a global coarse grid problem. This is then solved exactly by some unspecified algorithm, possibly of course a standard parallel multigrid method.

- The solution of the global coarsest grid problem is used by each processor to correct its local solution approximation on the second coarsest level. As in the following correction steps on the finer levels the values at the interfaces are included into the correction.

- After this step each processor finishes its V-cycle, again without communication with its neighbours.

The basic principle the algorithm uses to reduce communication is the following:

- Clearly some exchange of information between the processors is inevitable to solve the problem. In the algorithm by Brandt and Diskin however, this data exchange is restricted to the coarse grid correction from the common coarsest grid. So there is no communication on the finer grid levels.
To compensate errors introduced by the missing exchange of information, each subdomain has a buffer area around it, that fulfills two purposes. On the one hand, if an appropriate relaxation scheme, like e.g. red–black Gauß–Seidel is chosen, the buffer slows the propagation of errors due to wrong values at the interfaces. On the other hand, as in standard multigrid, the coarse grid correction introduces some high–frequency error components on the fine grid. Since the values at the interfaces cannot be smoothed, the algorithm cannot eliminate these components. But in elliptic problems high–frequency components decay quickly. So the buffer area keeps them from affecting the inner values too much.

Nevertheless the algorithm will in several cases not be able to produce an exact solution of the discrete problem. While this seems prohibitive at first glance, one should remember that, when solving a PDE, it is the continuous solution one really interested in. And the latter is represented by the discrete solution only up to the discretization error. This means, that as long as it can be guaranteed, that the algebraic error in the solution of the algorithm remains at least smaller than the discretization error, the solution is useful.

Since the overlap areas are included in the V–cycle, the algorithm trades communication for calculation. How much the time saving due to renunciation of communication on the finer grid levels will again be diminished through the additional calculation work depends on several factors, such as size of the problem, number and arrangement of subgrids, extension of the overlap areas and the MFLOP– and transfer–rates of the applied hard– and software. This will be further examined in the following section.

A crucial aspect of the above algorithm is of course the choice of an appropriate size for the overlap between subdomains. Until now there exists no general a priori error estimator, that would allow to give a bound for the algebraic error depending on the overlap parameter $J$. But it is possible to calculate safe margin lengths, that will result in a vanishing algebraic error. According to [4] for instance, one V–cycle of the algorithm with an overlap size of

\[ J(l) > 2^{1 + \nu/k} \left( 2k-1 - 1 \right) \nu \]

on the $l$th grid level will produce the exact discrete solution. Here $k$ is the total number of levels and $\nu$ is the sum of pre– and post–smoothing steps. Fortunately already much smaller overlaps will produce reasonable results, as can be seen from the numerical experiments in [4] and the further considerations therein.

### 3.2 Possible Time Saving

In this section we will explore how much overall computation time can be saved by the trade–off between communication and additional computation in the overlap areas. As mentioned in the previous section this depends on several architecture and problem parameters. To examine this question we compare a “standard” parallel V–cycle for the coarse grid correction (CGC) to an identical V–cycle that does not exchange data between processors, but instead employs overlapping subdomains.

We are going to model the times spent with calculation and communication within the algorithm from the following simplifying assumptions:

- Data exchange between processors is performed by message passing,
- Communication and calculation are sequential operations and cannot be overlapped.
• A processor can send and receive messages simultaneously.

We now define a relative time saving \( T_{\text{rs}} \) per grid level in the following way

\[
T_{\text{rs}} = \frac{T(\text{stand}) - T(\text{nocom})}{T(\text{stand})}.
\]

Here \( T(\text{nocom}) \) is the time for the variant without communication and \( T(\text{stand}) \) for the standard variant with communication. The times include all the work that has to be done for the specific grid level within one cycle of the CGC-scheme, that is smoothing, calculation of the defect, restriction of the defect, prolongation of the coarse grid solution and adding of the correction. They can be split in the following way

\[
T(\text{stand}) = T_{\text{calc}}(\text{stand}) + T_{\text{comm}}
\]

\[
T(\text{nocom}) = T_{\text{calc}}(\text{stand}) + T_{\text{calc}}(\text{buffer}).
\]

To determine the times for communication and calculation we use the following two models

\[
T_{\text{calc}} = \frac{\gamma}{n_p} \left( \nu F_{\text{smooth}} N_1 + F_{\text{multi}} N_2 \right)
\]

\[
T_{\text{comm}} = \frac{1}{n_p} \left( \alpha M + \beta W \right)
\]

with the accompanying parameters

| \( \alpha \) | latency |
| \( \beta \) | bandwidth |
| \( \gamma \) | Time per FLOP |
| \( n_p \) | no. of processors |
| \( \nu \) | sum of smoothing steps |

and

| \( F_{\text{smooth}} \) | no. of FLOPs per point and smoothing step |
| \( F_{\text{multi}} \) | no. of FLOPs per point for coarse grid correction |
| \( N_1 \) | no. of points that are smoothed |
| \( N_2 \) | no. of points that are included in the coarse grid correction |
| \( M \) | no. of messages that must be exchanged |
| \( W \) | no. of words (double precision values) that must be exchanged |

Here \( \alpha \) and \( \beta \) are the parameters already known from equation (1).

We consider the two-dimensional model problem from section 2.1, with a 5-point discretization, a 9-point stencil for restriction, bilinear interpolation and red-black Gauß-Seidel as smoother, and its three-dimensional analogue, where we use a 7-point discretization, a 27-point stencil for restriction and trilinear interpolation. We assume that our global grid is quadratic / cubic with \( N^d \) points along each border (\( d = 2 \) or 3) and that we have a logical processor grid. Assuming furthermore that NODD is employed for the grid partitioning, we can, for given values of \( N \), \( p_x \), \( p_y \) and eventually \( p_z \) and for a given overlap parameter \( J \), calculate the corresponding values of \( n_p \), \( N_1 \), \( N_2 \), \( M \) and \( W \). We estimate the number of numerical operations per grid point as follows:
Now we choose two different processor types, i.e. two different values for $\gamma$, one with 200 and the other one with 50 MFLOPS and examine the two-dimensional model problem. We compare the relative time saving $T_{res}$ for different grid sizes $N$ and different communication speeds $\beta$. Latency effects are taken into account by assuming that $\alpha = const \cdot \beta$. The results can be seen in figure 5. The analogous results for the three-dimensional model problem are shown in figure 7.

Communication hardware may have widely varying performance characteristics. In the following table we present typical parameters for current implementations of the message passing interface (MPI). The values have been taken from [5].

<table>
<thead>
<tr>
<th></th>
<th>Myrinet</th>
<th>Fast Ethernet</th>
<th>Ethernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>70$\mu$s</td>
<td>630$\mu$s</td>
<td>1150$\mu$s</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0.36$\mu$s</td>
<td>2$\mu$s</td>
<td>11.4$\mu$s</td>
</tr>
<tr>
<td>$\alpha/\beta$</td>
<td>194</td>
<td>315</td>
<td>100</td>
</tr>
</tbody>
</table>

The results in figures 5 and 7 confirm the following properties of the approach. The value $T_{res}$ of the relative time saving grows with $\beta$. This was to be expected, because the more expensive communication is, the greater the gain by replacing it with computation.

However, as the grids become larger $T_{res}$ is reduced. Although the margin between communication time $T_{comm}$ and additional calculation time $T_{calc}(\text{buffer})$ becomes more and more favourable as $N$ grows, this is to some extent compensated by the growing influence of the regular computational work $T_{calc}(\text{stand})$ on the overall time. This can clearly be seen by comparing the figures for the 200 and the 50 MFLOP case.

This compensation effect is significantly smaller in 3D than in 2D, since here the number of points and the amount of computation is of order $O(N^3)$, whereas the number of interface points, responsible for communication, grows like $N^2$. In 2D, on the other hand, these values are $N^2$ and $N$, respectively.

As far as the remaining problem parameters are concerned, some properties can easily be derived. If we consider the overlap parameter $J$, it can be said that, as long as $N \gg J$, the additional computation grows almost linearly with $J$. So we have that $T_{res} = C_1 - C_2 J$, with constants $C_k$ depending on the other parameters.

We found that the parameter $\nu$ had little influence on $T_{res}$, but the number of processors $n_p$ does. The relative time saving is more favourable, when there are more processors, because then there are fewer points on each subgrid and therefore the above mentioned compensation effect is reduced.

4 The Two Level Brandt–Diskin–Algorithm

In this section we will take a closer look at the two level version of the multilevel cycle in order to analyse in detail the communication pattern of the algorithm.

4.1 Description

Let us consider the model problem on the unit square $\Omega$. We will discretize the problem on a fine grid $\Omega^h := \{x_{ij} \mid 0 \leq i, j \leq N\}$ and a coarse grid $\Omega^H := \{x_{2i, 2j} \mid 0 \leq i, j \leq N/2\}$. 
Figure 5: Relative Time Saving for the 2D-Model Problem

Figure 6: Legend for the contourplots in figures 5 and 7

Figure 7: Relative Time Saving for the 3D-Model Problem
Here \( x_{ij} = (ih, jh) \) denotes a grid point on the \( i \)th vertical and the \( j \)th horizontal line. We decompose the fine grid into two equally large parts \( \Omega^{h,1} \) and \( \Omega^{h,2} \) with the grid line \( j = N/2 \) as common boundary. Two subdomains will suffice to explain the basic concept and the extension towards more subdomains will be straightforward. Now we augment each subgrid with a buffer zone of \( J > 0 \) grid lines and get the extended subdomains

\[
\hat{\Omega}^{h,1} := \{ x_{ij} \mid j \leq N/2 + J \} \quad \text{and} \quad \hat{\Omega}^{h,2} := \{ x_{ij} \mid j \geq N/2 - J \}.
\]

The quantity \( J > 0 \) describes the amount of overlap between the two subdomains. It is required that \( J \) is a multiple of 2 to ensure that there are coarse grid points that coincide with interface points on the fine grid.

Each subgrid is now assigned to one processor, \( p_1 \) and \( p_2 \). Starting from an approximation \( u_h^0 \) to the discrete solution of the fine grid problem the algorithm is defined by iteratively performing the following seven steps:

1. Both processors perform separate pre-smoothing steps using red-black Gauss-Seidel iteration.
2. A global approximate solution is composed from the local subdomain solutions.
3. A global defect function is composed.
4. With the global solution and defect the right hand side of the coarse grid equation is calculated according to the FAS-scheme.
5. The coarse grid equation is solved exactly (by whatever means).
6. Each processor uses the coarse grid solution to correct its local solution approximation.
7. Both processors perform separate post-smoothing steps.

In step 1 and 7 separate means that there is no communication between the processors to update values along the interfaces. As a consequence the local solutions \( u^{2,1} \) and \( u^{2,2} \) will start to differ.

The steps 2 and 3 are of virtual character in the sense that they are never performed explicitly, i.e., in a way that each processor knows the global functions completely. Formally the global solution is simply computed by taking the values of the local solutions in the interior of the subdomains \( \Omega^{2,p} \) and their average along their common boundary:

\[
\begin{align*}
 u^h(x_{ij}) := \begin{cases} 
 u^{h,1}(x_{ij}) & \text{if } j < \frac{N}{2} \\
 \frac{1}{2} \left( u^{h,1}(x_{ij}) + u^{h,2}(x_{ij}) \right) & \text{if } j = \frac{N}{2} \\
 u^{h,2}(x_{ij}) & \text{if } j > \frac{N}{2}
\end{cases}
\end{align*}
\]

We compose a global defect function in just the same way, namely as

\[
\begin{align*}
 r^h(x_{ij}) := \begin{cases} 
 r^{h,1}(x_{ij}) & \text{if } j < \frac{N}{2} \\
 \frac{1}{2} \left( r^{h,1}(x_{ij}) + r^{h,2}(x_{ij}) \right) & \text{if } j = \frac{N}{2} \\
 r^{h,2}(x_{ij}) & \text{if } j > \frac{N}{2}
\end{cases}
\end{align*}
\]

Here \( r^h_p = f^h - I^h u^h_p \) denotes the local defect functions and \( f^h \) and \( I^h \) are to be interpreted as restrictions on the extended subdomains \( \hat{\Omega}^h_p \).
We point out that in general the so defined global defect function \( r^h \) is different from the defect \( f^h - I^h u^h \) of the global solution function at points near the interface. We will return to this fact in section 4.3.

In a normal FAS-scheme the correction in step 6 would be performed as

\[
(19) \quad u^h_{\text{new}} = u^h_{\text{old}} + I^H_H \left[ u^H - I^H_H u^h_{\text{old}} \right]
\]

where \( u^H \) is the solution of the coarse grid problem obtained in step 5, and \( I^H_H \) and \( I^H_H \) are a prolongation and a restriction operator.

In the two level cycle basically the same is done, but since the global solution is not locally available to the processors, the local solution is corrected instead. To do this we define a prolongation operator

\[
(20) \quad J^h_{\text{g}} : \tilde{\Omega}^H \to \hat{\Omega}^h_{\text{p}}
\]

calculating the values in \( \hat{\Omega}^h_{\text{p}} \) from the respective values in \( \Omega^H \cap \hat{\Omega}^h_{\text{p}} \) by bilinear interpolation and a restriction operator

\[
(21) \quad J^H_{\text{h}} : \hat{\Omega}^h_{\text{p}} \to \Omega^H
\]

which computes the values in \( \Omega^H \cap \hat{\Omega}^h_{\text{p}} \) through injection and sets the values in \( \Omega^H \setminus \hat{\Omega}^h_{\text{p}} \) to zero. Now the correction can be written as

\[
(22) \quad u^h_{\text{new}} = u^h_{\text{old}} + J^h_{\text{g}} \left[ u^H - J^H_{\text{h}} u^h_{\text{old}} \right]
\]

### 4.2 Embedded Information Exchange

In the following we will explain, how the coarse grid correction from above leads to an information update between the two processors.

To see this, let us consider a point \( P = x_{ij} \) that lies in the overlap area \( \tilde{\Omega}^{h,1} \cap \tilde{\Omega}^{h,2} \). For such a point both processors store a local solution value. Because of the lack of communication during relaxation these values will in general differ prior to the correction step. If we denote

![Figure 8: A point with values at both processors](image)

by \( E^p \) the identity operator on \( \hat{\Omega}^h_{\text{p}} \) the value of the point after the correction (22) can be written as

\[
(23) \quad u^h_{\text{new}}(P) = \left( E^p - J^h_{\text{g}} J^H_{\text{h}} \right) u^h_{\text{old}}(P) + \left( J^h_{\text{g}} u^H \right)(P).
\]

The second term in the correction formula evaluates to the same value for both processors since the point lies in the overlap area. If our point has even indices \( P = x_{2i,2j} \) it can also be found on the coarse grid and therefore the first term in the correction formula will vanish independently of the particular processor. So we get that

\[
(24) \quad u^h_{\text{new}}(P) = \left( J^h_{\text{g}} u^H \right)(P) = u^H(P) = \left( J^h_{\text{g}} u^H \right)(P) = u^h_{\text{new}}(P)
\]
which means that after the correction step the values of the two processors at the point will coincide.

For the other points in the overlap area, those which cannot be found on the coarse grid, the first term will in general not vanish. This is due to interpolation errors. So the values of the two processors will be composed of a common part, namely $(J^h_{ij} u^H(x_{ij}))$, and a disturbance, which is a remainder of the old function value at each processor, and therefore will in general make the two values differ. So the coarse grid correction leads to a certain synchronisation between the local solutions stored by the different processors.

4.3 Properties

In the following we want to point out some of the features of the two level cycle. First of all, since the algorithm represents an iteration on the pair of local solution functions $(u^{h,1}_i, u^{h,2}_i)$, there are two obvious questions. Does the iteration converge and if it does, what is the limit it converges to.

To answer the first question: The algorithm converges for every choice of initial values. As far as the second question is concerned we have already pointed out that the iteration does not converge to the exact solution $u_{_{\text{disc}}}$ of the discrete problem in the sense that

$$\lim_{i \to \infty} u^{h,p}_i = u_{_{\text{disc}}}(\chi_{\Omega^p}).$$

In fact, the pair of limit functions is not even unique, but depends on the choice of initial values. This should not be surprising, since the exact solution on a subdomain solely depends on the boundary values. While the outer boundary values are given by the problem, the interface values vary from step to step as they are changed by the correction.

We imagine now, that we split the function of the interface values into two parts. The first part is determined by replacing the value at every second point by the linear interpolation from the neighbouring points. The second part, the so called hierarchical offset, is then given as the difference between the original function and the first part.

With this in mind, we note the following. The first term in (23) will always reproduce the hierarchical offset of the solution. The second term on the other hand represents a bilinear interpolation and therefore has no hierarchical offset at all. As a consequence the hierarchical offset of the initial values along the inner boundaries is never changed by the algorithm. Our experiments confirm that the initial values can be grouped in equivalence classes according to their hierarchical offset, with every class converging to the same limit function.

Another property that we demand of the two level cycle is that for a proper choice of the overlap parameter $J$ the algebraic error of the algorithm (the error with respect to the true discrete solution), remains smaller or of the same order of magnitude as the discretization error. The numerical experiments in part 5 will show that the two level cycle possesses this feature and that it can already be achieved with an overlap of 2 to 4 grid lines.

This property can be improved, by a different way to compose a global defect function $r^h$. We have already mentioned that, if the global defect function is set up according to (18) it does not match the defect $r^h := f^h - L^h u^h$ of the global solution approximation. If we use $r^h$ as the global defect, the algebraic error will be smaller than with the use of $r^h$ and it will strongly be restricted to the vicinity of the common boundary of the subdomains. Also we then need only smaller overlaps to keep the algebraic error smaller than the discretization error and this will often be achieved in a smaller number of cycles.

An argument that might be held against the use of $r^h$ is, that it needs more communication to set up the coarse grid equation. The total data exchange to do this depends in general
on the method employed for solving the coarse grid equation, e.g., a sequential direct solver or a parallel solver, like standard parallel multigrid. But since the calculation of the defect $f^h - L^h u^h$ as well as the prolongation by full weighting and the application of the coarse grid operator $L^H$ are linear operations the amount of communication needed to set up the coarse grid equation is the same if we use $f^h$ or $f^H$.

A more detailed argument is now presented for a special case. Assume we use an exact solver instead of the smoothing in steps 1 and 7 of the algorithm and choose initial values without a hierarchical offset at the inner boundaries. Then the algorithm becomes equivalent to two coupled Alternating Schwarz Methods (ASM), see e.g., [8] or [14], that use linear interpolation in the coupling of the subdomains. This is schematically shown in figure 9. There $R_y^{(k)}$ and $u_i^{(k)}$ denote the interface values at the boundary of $\Omega_y^h$ and the approximate solution on that subdomain after the $k$th half-cycle. In [11] we have shown that the Alternating Schwarz Method with interpolation represents a convergent iterative process, whose limit is independent of the initial values, as is the two level cycle for the equivalence class with zero hierarchical offset in the initial values. The analytical and numerical results for this special case confirm our considerations and tests for the general case.

5 Numerical Experiments

In this section we will give some numerical results for the two level cycle. In order to demonstrate the working of the algorithm and the nature of its algebraic error we start with a simple problem.

Let $\Delta u = 0$ on $\Omega$ and $u = 0$ on $\partial \Omega$ where $\Omega = (0,1)^2$ is the unit square. We discretize the domain with $33 \times 33$ points and divide it into two equally large subdomains with the border parallel to the $x$-axis. Each subdomain is then enlarged by a buffer area of $J = 4$ grid lines. As initial values we choose $u(x,y) = \sin(\pi x)\sin(\pi y)$ and perform $\nu = 2$ pre- and post-smoothing steps each cycle.

In figure 10 we compare the results of the two level cycle for the original version that uses an averaged global defect according to (18) with the modified one that uses the true global defect $r^h = f^h - L^h u^h$. After the first cycle there are no visible differences in the approximate solutions $u^h$. The reason for this is, that, since we employ red-black Gauss-Seidel as smoother, the error from the interfaces has not reached the inner points after the pre-smoothing steps and the difference in the defects is still minimal. This changes in the following cycles when the results start to deviate from each other and also from the way a “standard” multigrid method would behave. Typical for the original version is the widely spread algebraic error in the solution. In the modified version the error is smaller and concentrated along the interface.
Figure 10: Results of the original (left) and the modified (right) version of the two level cycle.
Comparing the results for the original version after the second and the third cycle one notices that the error has again increased. This may easily happen since the limit solution and the exact solution do not coincide.

In the following we will compare both versions of the method by means of the Poisson problem with the exact solution

\[
\tag{26}
u(x,y) = \cos \left[ A (x + 4) + B (y + 4) \right]
\]

on the domain \( \Omega = (-4, 4)^2 \). We will use the following three pairs of parameters \((A, B)\) that were already used in [3] and [4] to test the original version:

\(\begin{align*}
(i) & \quad A = 4 \pi, \quad B = \frac{\pi}{2} \\
(ii) & \quad A = \frac{\pi}{2}, \quad B = 4 \pi \\
(iii) & \quad A = \frac{\pi}{2}, \quad B = \frac{\pi}{2}
\end{align*}\)

Discretizing the domain with a quadratic grid and choosing the border between our two subdomains to be along the \(x\)-axis, this gives us three problems. One with a high oscillation parallel (i) and one with a high oscillation perpendicular (ii) to the interface and a low oscillation in the other direction, as well as a problem with a smooth solution in both directions (iii).

We compare the solutions of the two algorithmic versions for the three parameter settings (i) – (iii) and for different mesh sizes \( h = 1/N \) and overlap parameters \( J \). Initial values are calculated by solving an associated coarse grid problem and bicubic interpolation of the resulting solution to the fine grid. In each cycle one pre- and one post-smoothing step is performed. The results are given in tables 2 to 4. Comparing them one notes the following:

- In both norms the algebraic error of the modified version is smaller than that of the original version, indicating that the error of the former is more concentrated along the interfaces. This can also be seen in picture 11, that shows a typical algebraic error.

![Figure 11: Algebraic error of the limit solution of the original (left) and the modified version (right) of the two level cycle for problem (26) with parameters \((4\pi, \frac{\pi}{2})\) and \(J = 4\)]
<table>
<thead>
<tr>
<th>row</th>
<th>content of row</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>a fine grid of $(N + 1) \times (N + 1)$ points is used</td>
</tr>
<tr>
<td>$J$</td>
<td>each buffer area consists of $J$ grid lines</td>
</tr>
<tr>
<td>Norm</td>
<td>$L_2$ stands for the discrete $L_2$-norm and $\max$ for the maximum-norm</td>
</tr>
<tr>
<td>Res</td>
<td>for old the global defect was calculated with the original method (18), while for new the modified version was used</td>
</tr>
<tr>
<td>disc. err.</td>
<td>norm of discretization error</td>
</tr>
<tr>
<td>$0$</td>
<td>norm of algebraic error in the initial values</td>
</tr>
<tr>
<td>$1 - 3$</td>
<td>norm of algebraic error of the approximate solution at the end of the respective cycle</td>
</tr>
<tr>
<td>$\infty$</td>
<td>norm of the algebraic error in the limit solution</td>
</tr>
</tbody>
</table>

Table 1: Legend for tables 2 to 4

- In practically all cases an overlap of $J = 2$ was sufficient to make the algebraic error smaller in magnitude than the discretization error.
- The larger the extension of the buffer areas is, the more cycles it takes before differences in the results of the two versions show up. This is the same effect as explained in the first example.
- While the decrease in the error is monotone with respect to $J$ for the problems (ii) and (iii), this is not the case for problem (i). Here a local minimum seems to exist for $J = 4$. We also noticed this effect in the analysis of the coupled Alternating Schwarz Methods, cf. section 4 and [11].

As third example we take the modified algorithm and examine the Poisson–Equation on the unit square $(0,1)^2$ with exact solution

\[
u(x,y) = \frac{f(x)f(y) - 1}{e^{20} - 1}, \quad f(t) = e^{10 \sin(5 \pi t)}\]

<table>
<thead>
<tr>
<th>N</th>
<th>J</th>
<th>Norm</th>
<th>Res</th>
<th>disc. err.</th>
<th>$0$</th>
<th>$1$</th>
<th>$2$</th>
<th>$3$</th>
<th>$\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>2</td>
<td>$L_2$</td>
<td>old</td>
<td>7.83 \cdot 10^{-6}</td>
<td>1.40 \cdot 10^{-1}</td>
<td>3.64 \cdot 10^{-1}</td>
<td>1.46 \cdot 10^{-1}</td>
<td>1.44 \cdot 10^{-1}</td>
<td>1.43 \cdot 10^{-1}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>new</td>
<td></td>
<td>5.17 \cdot 10^{-1}</td>
<td>3.95 \cdot 10^{-1}</td>
<td>3.96 \cdot 10^{-1}</td>
<td>3.96 \cdot 10^{-1}</td>
<td>3.96 \cdot 10^{-1}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>old</td>
<td>2.39 \cdot 10^{-6}</td>
<td>4.11 \cdot 10^{-10}</td>
<td>1.12 \cdot 10^{-10}</td>
<td>4.67 \cdot 10^{-10}</td>
<td>4.65 \cdot 10^{-10}</td>
<td>4.61 \cdot 10^{-10}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>new</td>
<td>3.91 \cdot 10^{-10}</td>
<td>3.75 \cdot 10^{-10}</td>
<td>3.79 \cdot 10^{-10}</td>
<td>3.79 \cdot 10^{-10}</td>
<td>3.79 \cdot 10^{-10}</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>$L_2$</td>
<td>old</td>
<td>7.83 \cdot 10^{-6}</td>
<td>1.40 \cdot 10^{-1}</td>
<td>4.11 \cdot 10^{-1}</td>
<td>2.00 \cdot 10^{-2}</td>
<td>1.75 \cdot 10^{-3}</td>
<td>1.26 \cdot 10^{-3}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>new</td>
<td></td>
<td>4.11 \cdot 10^{-1}</td>
<td>2.00 \cdot 10^{-2}</td>
<td>1.12 \cdot 10^{-3}</td>
<td>3.77 \cdot 10^{-6}</td>
<td>3.77 \cdot 10^{-6}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>old</td>
<td>2.39 \cdot 10^{-6}</td>
<td>4.11 \cdot 10^{-10}</td>
<td>2.30 \cdot 10^{-10}</td>
<td>1.23 \cdot 10^{-10}</td>
<td>7.05 \cdot 10^{-10}</td>
<td>4.16 \cdot 10^{-10}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>new</td>
<td>2.30 \cdot 10^{-10}</td>
<td>1.23 \cdot 10^{-10}</td>
<td>6.66 \cdot 10^{-10}</td>
<td>2.86 \cdot 10^{-10}</td>
<td>2.86 \cdot 10^{-10}</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>6</td>
<td>$L_2$</td>
<td>old</td>
<td>7.83 \cdot 10^{-6}</td>
<td>1.40 \cdot 10^{-1}</td>
<td>4.11 \cdot 10^{-1}</td>
<td>2.19 \cdot 10^{-2}</td>
<td>9.47 \cdot 10^{-2}</td>
<td>9.91 \cdot 10^{-2}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>new</td>
<td></td>
<td>4.11 \cdot 10^{-1}</td>
<td>2.00 \cdot 10^{-2}</td>
<td>1.56 \cdot 10^{-3}</td>
<td>1.13 \cdot 10^{-3}</td>
<td>1.13 \cdot 10^{-3}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>old</td>
<td>2.39 \cdot 10^{-6}</td>
<td>4.11 \cdot 10^{-10}</td>
<td>2.30 \cdot 10^{-10}</td>
<td>1.24 \cdot 10^{-10}</td>
<td>2.88 \cdot 10^{-10}</td>
<td>3.02 \cdot 10^{-10}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>new</td>
<td>2.30 \cdot 10^{-10}</td>
<td>1.23 \cdot 10^{-10}</td>
<td>1.32 \cdot 10^{-10}</td>
<td>9.43 \cdot 10^{-10}</td>
<td>9.43 \cdot 10^{-10}</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Results for problem (26) with parameters $(4\pi, \frac{5}{2})$
<table>
<thead>
<tr>
<th>N</th>
<th>J</th>
<th>Norm</th>
<th>Res</th>
<th>disc. err.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>∞</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>2</td>
<td>$L_2$</td>
<td>old new</td>
<td>7.83 \times 10^{-6}</td>
<td>1.38 \times 10^{+1}</td>
<td>8.45 \times 10^{-1}</td>
<td>1.02 \times 10^{-1}</td>
<td>8.13 \times 10^{-2}</td>
<td>8.48 \times 10^{-2}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>old new</td>
<td>2.30 \times 10^{-6}</td>
<td>4.11 \times 10^{+0}</td>
<td>3.68 \times 10^{-1}</td>
<td>6.49 \times 10^{-2}</td>
<td>7.17 \times 10^{-2}</td>
<td>7.17 \times 10^{-2}</td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>$L_2$</td>
<td>old new</td>
<td>7.83 \times 10^{-6}</td>
<td>1.38 \times 10^{+1}</td>
<td>4.11 \times 10^{-1}</td>
<td>1.29 \times 10^{-1}</td>
<td>1.29 \times 10^{-2}</td>
<td>1.29 \times 10^{-2}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>old new</td>
<td>2.30 \times 10^{-6}</td>
<td>4.11 \times 10^{+0}</td>
<td>2.30 \times 10^{-1}</td>
<td>6.57 \times 10^{-2}</td>
<td>6.46 \times 10^{-3}</td>
<td>7.36 \times 10^{-3}</td>
</tr>
<tr>
<td>32</td>
<td>6</td>
<td>$L_2$</td>
<td>old new</td>
<td>7.83 \times 10^{-6}</td>
<td>1.38 \times 10^{+1}</td>
<td>4.11 \times 10^{-1}</td>
<td>2.00 \times 10^{-2}</td>
<td>1.30 \times 10^{-3}</td>
<td>5.12 \times 10^{-4}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>old new</td>
<td>2.30 \times 10^{-6}</td>
<td>4.11 \times 10^{+0}</td>
<td>2.30 \times 10^{-1}</td>
<td>1.29 \times 10^{-2}</td>
<td>6.71 \times 10^{-4}</td>
<td>3.33 \times 10^{-4}</td>
</tr>
</tbody>
</table>

Table 3: Results for problem (26) with parameters ($\frac{\pi}{2}$, $4\pi$)

<table>
<thead>
<tr>
<th>N</th>
<th>J</th>
<th>Norm</th>
<th>Res</th>
<th>disc. err.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>∞</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>2</td>
<td>$L_2$</td>
<td>old new</td>
<td>1.70 \times 10^{-2}</td>
<td>5.38 \times 10^{+2}</td>
<td>9.31 \times 10^{-4}</td>
<td>7.52 \times 10^{-4}</td>
<td>7.57 \times 10^{-4}</td>
<td>7.34 \times 10^{-4}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>old new</td>
<td>4.00 \times 10^{-3}</td>
<td>1.31 \times 10^{+2}</td>
<td>2.52 \times 10^{-4}</td>
<td>4.90 \times 10^{-4}</td>
<td>1.60 \times 10^{-5}</td>
<td>4.83 \times 10^{-4}</td>
</tr>
<tr>
<td>128</td>
<td>2</td>
<td>$L_2$</td>
<td>old new</td>
<td>4.25 \times 10^{-3}</td>
<td>1.29 \times 10^{+0}</td>
<td>6.34 \times 10^{-5}</td>
<td>9.76 \times 10^{-5}</td>
<td>9.69 \times 10^{-5}</td>
<td>9.66 \times 10^{-5}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>old new</td>
<td>9.97 \times 10^{-4}</td>
<td>3.06 \times 10^{+3}</td>
<td>1.48 \times 10^{-5}</td>
<td>6.49 \times 10^{-5}</td>
<td>6.50 \times 10^{-5}</td>
<td>6.48 \times 10^{-5}</td>
</tr>
<tr>
<td>128</td>
<td>4</td>
<td>$L_2$</td>
<td>old new</td>
<td>4.25 \times 10^{-3}</td>
<td>1.29 \times 10^{+0}</td>
<td>6.34 \times 10^{-5}</td>
<td>1.26 \times 10^{-5}</td>
<td>1.27 \times 10^{-5}</td>
<td>1.27 \times 10^{-5}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>old new</td>
<td>9.97 \times 10^{-4}</td>
<td>3.06 \times 10^{+3}</td>
<td>1.47 \times 10^{-5}</td>
<td>7.77 \times 10^{-6}</td>
<td>1.00 \times 10^{-5}</td>
<td>1.00 \times 10^{-5}</td>
</tr>
<tr>
<td>128</td>
<td>8</td>
<td>$L_2$</td>
<td>old new</td>
<td>4.25 \times 10^{-3}</td>
<td>1.29 \times 10^{+0}</td>
<td>6.34 \times 10^{-5}</td>
<td>4.83 \times 10^{-7}</td>
<td>2.75 \times 10^{-8}</td>
<td>1.50 \times 10^{-8}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>old new</td>
<td>9.97 \times 10^{-4}</td>
<td>3.06 \times 10^{+3}</td>
<td>1.47 \times 10^{-5}</td>
<td>5.61 \times 10^{-7}</td>
<td>1.88 \times 10^{-7}</td>
<td>1.83 \times 10^{-7}</td>
</tr>
<tr>
<td>128</td>
<td>16</td>
<td>$L_2$</td>
<td>old new</td>
<td>4.25 \times 10^{-3}</td>
<td>1.29 \times 10^{+0}</td>
<td>6.34 \times 10^{-5}</td>
<td>4.83 \times 10^{-7}</td>
<td>1.70 \times 10^{-8}</td>
<td>1.77 \times 10^{-10}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>old new</td>
<td>9.97 \times 10^{-4}</td>
<td>3.06 \times 10^{+3}</td>
<td>1.47 \times 10^{-5}</td>
<td>5.61 \times 10^{-7}</td>
<td>2.63 \times 10^{-8}</td>
<td>9.40 \times 10^{-9}</td>
</tr>
</tbody>
</table>

Table 4: Results for problem (26) with parameters ($\frac{\pi}{2}$, $\frac{\pi}{2}$)
whose graphical representation can be seen in figure 12. Our aim here is twofold. First, we want to see how the algebraic error behaves, when the mesh size $h = 1/N$ is decreased while the overlap parameter $J$ is kept constant. To see this, we choose initial values the same way like in the before mentioned problem, fix $J = 2$ and increase $N$ from 32 to 1024. Table 5 gives the norms of the algebraic errors after the first cycles and for the limit solution. We see here as we do in figure 13, that the algebraic error decreases faster than the discretization error for this example and that an overlap of $J = 2$ grid lines is more than adequate.

To confirm the property, that the limit solution is identical for all initial values that belong into the same equivalence class with respect to the hierarchical surplus at the interfaces, we then recalculated the problem with different initial values. Those were computed by taking the old initial values at the interfaces and random values between -1 and 1 at all other points. The modified algorithm converged to the same limit solution in every case and the speed of convergence was not noticeably impaired either.

### Table 5: Development of the discretization and algebraic error for problem (27) with growing $N$ and fixed $J = 2$

<table>
<thead>
<tr>
<th>$N$</th>
<th>Norm</th>
<th>disc. err.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>$\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Max</td>
<td>6.22×10^{-01}</td>
<td>1.20×10^{-00}</td>
<td>2.99×10^{-02}</td>
<td>4.92×10^{-03}</td>
<td>4.58×10^{-03}</td>
<td>4.57×10^{-03}</td>
</tr>
<tr>
<td></td>
<td>$L_2$</td>
<td>5.88×10^{-02}</td>
<td>1.14×10^{-01}</td>
<td>5.80×10^{-03}</td>
<td>4.42×10^{-04}</td>
<td>2.89×10^{-04}</td>
<td>2.86×10^{-04}</td>
</tr>
<tr>
<td>64</td>
<td>Max</td>
<td>8.39×10^{-02}</td>
<td>2.43×10^{-01}</td>
<td>1.78×10^{-02}</td>
<td>3.38×10^{-03}</td>
<td>3.05×10^{-03}</td>
<td>3.04×10^{-03}</td>
</tr>
<tr>
<td></td>
<td>$L_2$</td>
<td>6.28×10^{-03}</td>
<td>1.78×10^{-02}</td>
<td>1.31×10^{-03}</td>
<td>1.35×10^{-04}</td>
<td>1.02×10^{-04}</td>
<td>1.01×10^{-04}</td>
</tr>
<tr>
<td>128</td>
<td>Max</td>
<td>1.04×10^{-02}</td>
<td>5.94×10^{-02}</td>
<td>3.75×10^{-03}</td>
<td>1.69×10^{-03}</td>
<td>1.59×10^{-03}</td>
<td>1.59×10^{-03}</td>
</tr>
<tr>
<td></td>
<td>$L_2$</td>
<td>1.48×10^{-03}</td>
<td>3.12×10^{-03}</td>
<td>2.17×10^{-04}</td>
<td>3.76×10^{-05}</td>
<td>3.35×10^{-05}</td>
<td>3.33×10^{-05}</td>
</tr>
<tr>
<td>256</td>
<td>Max</td>
<td>4.75×10^{-03}</td>
<td>7.55×10^{-03}</td>
<td>3.34×10^{-04}</td>
<td>1.70×10^{-04}</td>
<td>1.66×10^{-04}</td>
<td>1.66×10^{-04}</td>
</tr>
<tr>
<td></td>
<td>$L_2$</td>
<td>3.65×10^{-04}</td>
<td>5.00×10^{-04}</td>
<td>1.41×10^{-05}</td>
<td>2.53×10^{-06}</td>
<td>2.42×10^{-06}</td>
<td>2.41×10^{-06}</td>
</tr>
<tr>
<td>512</td>
<td>Max</td>
<td>1.18×10^{-03}</td>
<td>1.37×10^{-03}</td>
<td>2.24×10^{-06}</td>
<td>1.20×10^{-05}</td>
<td>1.19×10^{-05}</td>
<td>1.19×10^{-05}</td>
</tr>
<tr>
<td></td>
<td>$L_2$</td>
<td>9.06×10^{-05}</td>
<td>1.11×10^{-04}</td>
<td>8.31×10^{-07}</td>
<td>1.15×10^{-07}</td>
<td>1.14×10^{-07}</td>
<td>1.14×10^{-07}</td>
</tr>
<tr>
<td>1024</td>
<td>Max</td>
<td>2.95×10^{-04}</td>
<td>3.04×10^{-04}</td>
<td>1.4×10^{-06}</td>
<td>7.48×10^{-07}</td>
<td>7.45×10^{-07}</td>
<td>7.45×10^{-07}</td>
</tr>
<tr>
<td></td>
<td>$L_2$</td>
<td>2.27×10^{-05}</td>
<td>2.68×10^{-06}</td>
<td>5.67×10^{-08}</td>
<td>4.57×10^{-09}</td>
<td>4.56×10^{-09}</td>
<td>4.56×10^{-09}</td>
</tr>
</tbody>
</table>
6 Conclusions

In this paper we have shown that the approach by Brandt and Diskin to parallelization of multigrid methods that trades computation for communication has some potential for reducing overall computation time. This remains valid even for high-speed communication infrastructures as long as the processor speeds are fast enough.

We have given in some ways an analysis of the two-level version of their algorithm that led to an improvement of its performance.

Open questions are a priori estimates for the algebraic error of the algorithm as well as convergence rates.

Another topic discussed in this paper was the application of non-overlapping domain decomposition techniques for parallel multigrid methods for finite difference problems on structured grids, that we showed to be favourable with respect to communication in some cases and to ease implementation.

References


REFERENCES


