On the Implementation of Algebraic Multi-level Methods on Vector and Parallel-Vector Computers

Lutz Grosz

School of Mathematical Science, Australian National University, Canberra, ACT 0200, Australia.

1. The Introduction

Iterative methods of the conjugate gradient type (CG, see [9]) are applied to solve the system of linear equations

$$Au = b \tag{1.1}$$

where $A = (a_{i,j})_{i,j=,1\ldots,n} \in \mathsf{R}^{n\times n}$ is the given, non-singular and sparse coefficient matrix, $b \in \mathsf{R}^n$ the given right hand side and $u \in \mathsf{R}^n$ the sought solution. In order to improve the convergence rate and the robustness of the iteration procedure a non–singular preconditioner matrix $M \in \mathsf{R}^{n\times n}$ is introduced. The CG method is applied to the transformed linear system:

$$M^{-1}Au = M^{-1}b \tag{1.2}$$

In order to limit the number of iteration steps the preconditioner matrix M has to be selected such that the eigenvalues of the new iteration matrix $M^{-1}A$ are clustered. Moreover the complexity for the evaluation of $M^{-1}p$ for a given vector $p \in \mathsf{R}^n$ must have moderate complexity in order to reduce significantly the computational effort for the whole solution procedure. Unfortunately in practical applications a preconditioner which fulfils these conditions in theory does not necessarily reduce the all–over computing time if the preconditioner cannot be or is not implemented efficiently on the used computer architecture.

The paper is aimed at the implementation of a subroutine which provides a robust and highly efficient solver for a linear system (1.1) which arises from the finite difference discretisation of a scalar boundary value problem of order two on a rectangular grid. The obvious preconditioner for this problem class is the multi–grid technique, see [4]. In order to achieve a reasonable flexibility and keep the interface of the subroutine simple for user the algebraic multi–level iteration (AMLI, see [1]) is the method of the choice. The method is related and competitive to the multi–grid method but is more robust and offers more flexibility regarding the problem, the discretisation method and the grid. Therefore AMLI needs only the coefficient matrix A as input, a fact that makes the application very handy for the user. However, it is clear that for some applications a suitable multi–grid solver is much faster than AMLI. This price has to be paid for more flexibility and robustness.

The target platform is a distributed memory, MIMD computer with shared memory, SIMD nodes (eg. Fujitsu VPP, NEC SX-5). The nodes are linked with a X-bar network with a high bandwidth ($1-4$ floats per cycle) but with a high latency ($\gg 1000$ cycles). Consequently on these architectures the transfer of one single long message has to be preferred the transfers of several shorter messages even if the total volume of the sent data is larger. This fact has to be considered in the distribution of the data over the nodes.

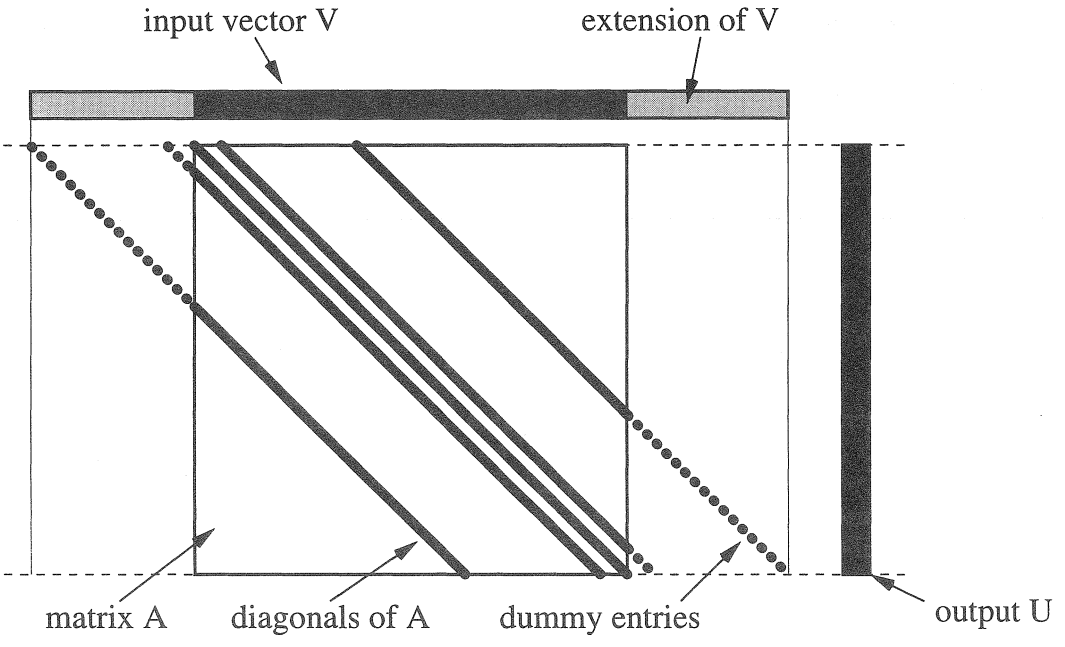

FIGURE 1. Vectorised matrix-vector multiplication.

The performance of a single node is quite impressive (> 2 Gflops). Unfortunately the start-up times for parallel operations (eg. vectorisable loops) is quite large. It is very surprising that for vector computers the start-up of vector instructions became even more costly during the last years (eg. > 300 cycles for a daxpy-operation an the Fujitsu VPP300). The reason for this trend is the introduction of parallel vector pipes which are delivering several results per cycle (eg. 16 results per cycle for NEC SX-5). Thus the usage of extremely long vectors is vital in order to get a good performance, eg. on the Fujitsu VPP300 the vector length for a daxpy-operation has to be larger than 3000 to achieve at least 90% of the possible peak performance. This is the reason why in the presented implementation vector oriented data structures are preferred to grid oriented data structure (as typically used in multi–grid codes).

In the first part of the paper the basic data structures of the implementation is presented. In the second part the AMLI algorithm is introduced as a block incomplete factorisation preconditioner. The third part of the paper discusses the approaches to get a highly efficient implementation on the relevant computer architectures as well as a black-box type routine which requires only the coefficient matrix as input data. It is emphasised that the selected methods require a logically rectangular grid, which can be recovered from the given coefficient matrix, but no information on the location of the grid points is needed. Some examples are presented and discussed in the fourth part.

2. Diagonal Storage Scheme

It is assumed the coefficient matrix A arises from the discretisation of a boundary value problem on a rectangular, d-dimensional grid. For $i = 1, \ldots, d$ m_i denotes the number of

grid points in the i-th spatial direction. Thus the number of unknowns is

$$n := \prod_{i=1}^{d} m_i \ . \tag{2.1}$$

The non–zero entries in the coefficient matrix A are concentrated in a few diagonals $\Delta = \{d_s | s = 1, \ldots, \kappa(\Delta)\} \subset \mathsf{Z}$, ie.

$$a_{i,j} \neq 0 \Rightarrow j - i \in \Delta \tag{2.2}$$

for all $i, j = 1, \ldots, n$. Notice that the i-th spatial direction is represented by the diagonal

$$D_i := \prod_{k=1}^{i-1} m_k \in \Delta \ , \tag{2.3}$$

for $i = 1, \ldots, d$, see Figure 1. In the case of a $2d+1$-point stencil used in a finite difference scheme of order two it is $\Delta = \{\pm D_i | i = 1, \ldots, d\} \cup \{0\}$. Thus it is $\kappa(\Delta) = 2d + 1$. Conversely it is simple to find a set $\{D_i | i = 1, \ldots, d\}$ in a given set of diagonals Δ that allows to recover the spatial dimension and the number of grid points in the spatial directions (especially in the case of 3^d-point stencil).

To get an efficient implementation of the CG iteration an optimal implementation of the matrix-vector multiplication has to be used as this operation consumes most computing time within one iteration step (especially for CG methods with short recurrences like classical CG and TFQMR, see [9]).

To get an efficient matrix-vector multiplication on a vector computer the matrix A is stored into the two-dimensional array $\mathtt{MAT}(n, \kappa(\Delta))$ using the diagonal storage scheme, see [8]: For diagonal $d_s \in \Delta$ with $d_s \geq 0$ the entries $(a_{i,i+d_s})_{i=1,\ldots,n-d_s}$ are stored into $\mathtt{MAT}(1 : n - d_s, s)$ where the entries $\mathtt{MAT}(n - d_s + 1 : n, s)$ are set to zero. For diagonals $d_s \in \Delta$ with $d_s < 0$ the entries $(a_{i,i+d_s})_{i=1-d_s,n}$ are stored into $\mathtt{MAT}(1 - d_s : n, s)$ where the entries $\mathtt{MAT}(1 : -d_s, s)$ are set to zero.

The matrix-vector multiplication $u := u + Av$ is executed by the following nested loops:

$$\begin{array}{l} \text{do } s = 1, \kappa(\Delta) \\ \quad \text{do } i = 1, n \\ \quad\quad \mathtt{U}(i) = \mathtt{U}(i) + \mathtt{MAT}(i, s) * \mathtt{V}(d_s + i) \\ \quad \text{end do} \\ \text{end do} \end{array} \tag{2.4}$$

where $\mathtt{U}$ is the array of the result vector u. Notice that the array $\mathtt{V}$ of the input vector $v \in \mathsf{R}^n$ must be allocated as $V(1 - B_\Delta : n + B_\Delta)$ where

$$B_\Delta = \max_{s=1}^{\kappa(\Delta)} |d_s| \tag{2.5}$$

is the bandwidth of the matrix A. The values $V(1 - B_\Delta : 0)$ and $V(n + 1 : n + B_\Delta)$ do not have any effect on the result, as the corresponding entries in the matrix array $\mathtt{MAT}$ are set to zero.

The inner loop of the procedure (2.4) can be vectorised with a very long vector length n. To avoid unnecessary load and store operations on the output vector $\mathtt{U}$ unrolling over the outer loop s can be introduced. This is possible as the length of the inner loop is independent from the outer loop by filling the array $\mathtt{MAT}$ with zeroes. The loop unrolling reduces the performance loss by the narrow memory bottleneck between memory and vector unit as it is common for recent vector architectures.

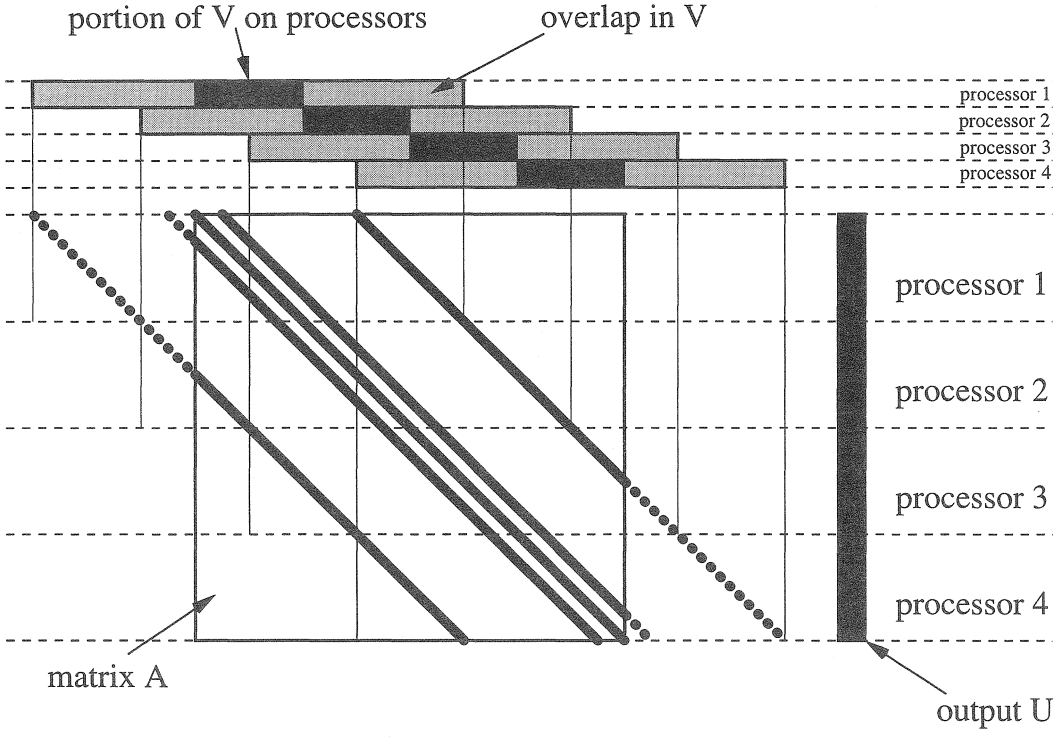

FIGURE 2. Parallel-vectorized matrix-vector multiplication.

3. Parallelisation

On a parallel computer with p nodes the array `U` as well as the array of the diagonals `MAT` are stored in banded form, ie. each node holds (about) $n_p := \frac{n}{p}$, consecutive entries of the arrays. For instance the array entries $\mathtt{U}((q-1)n_p+1 : qn_p)$ are stored on node $q \in \{1,\ldots,p\}$. The input vector `V` is stored with overlap, ie. $\mathtt{V}((q-1)n_p+1-B_\Delta : qn_p+B_\Delta)$ is stored on node $q \in \{1,\ldots,p\}$, see Figure 2. The entries in the overlap regions $\mathtt{V}((q-1)n_p+1-B_\Delta : (q-1)n_p)$ and $\mathtt{V}(qn_p+1 : qn_p+B_\Delta)$ have to be adjusted before starting the procedure (2.4). It is emphasised that the vector length for a parallel execution is reduced to n_p which limits the reasonable problem size for a particular number of nodes.

Notice that this distribution of the data corresponds to a slice–partioning of the underlying grid. Although a box–partioning would reduce the volume of transfered data to adjust the overlap the number of start-ups which are extremely costly on the relevant computer architectures is much higher than for a slice–partioning.

This parallelisation strategy, ie. partioning of the vectorised loop and data partioning with overlap (where necessary) is applied when implementing the algorithms presented in this paper.

4. Block Incomplete Factorisation

In this section we present the framework of block incomplete factorisation in order to construct the preconditioner M, see [7]:

Let C be a subset of the set of unknowns $\{1, 2, \ldots, n\}$. The n^C unknowns in C are called the coarse level unknowns and the unknowns in $F := \{1, 2, \ldots, n\} \setminus C$ are called the removed unknowns. The matrix A is subdivided in the following way:

$$A = \begin{bmatrix} A_{FF} & A_{FC} \\ A_{CF} & A_{CC} \end{bmatrix} . \tag{4.1}$$

The columns and rows of the sub-matrix $A_{FF} \in \mathsf{R}^{(n-n^C)\times(n-n^C)}$ belong to the removed unknowns in F and those of the sub-matrix $A_{CC} \in \mathsf{R}^{n^C \times n^C}$ to the coarse level unknowns in C.

Using a suitable definition of the coarse level unknowns A_{FF} is a diagonal matrix. To be more general it is assumed that A_{FF} has entries outside its main diagonal but can be approximated with reasonable accuracy by a diagonal matrix V_{FF}, eg. by using approximate inverse techniques. Notice that the usage of an approximation of A_{FF} allows the construction of a larger set of removed unknowns F which produces a smaller and thus faster solvable coarse level system.

Using the approximation V_{FF} of A_{FF} the approximate Schur complement

$$A^C := A_{CC} - A_{CF} V_{FF} A_{FC} \tag{4.2}$$

of A is calculated. The matrix $A^C \in \mathsf{R}^{n^C \times n^C}$ defines the coefficient matrix for the coarse level system. A block incomplete factorisation M of A is now constructed by

$$M = \begin{bmatrix} V_{FF}^{-1} & 0 \\ A_{CF} & I \end{bmatrix} \begin{bmatrix} I & V_{FF} A_{FC} \\ 0 & M^C \end{bmatrix} \tag{4.3}$$

where M^C is an approximation of the Schur complement A^C. If the dimension of A^C is reasonably large M^C is constructed by a block incomplete factorisation again. For 'small' matrices A^C the matrix M^C is implicitly constructed by the iterative solution of a linear system with coefficient matrix A^C. Normally an accuracy of three digits is sufficient. It is emphasised that in a strict sense the preconditioner is not a linear operator. The construction of M^C may include dropping strategies which is not considered in this paper, see [7]

The algorithm to calculate $p := M^{-1}q = \mathrm{BIF}(0, q)$ is the following recursive procedure:

```
1:  BIF(k, q)
2:    if (k = l)
3:      solve Ap = q
4:    else
5:      (q_F, q_C) ← q
6:      q_C ← q_C − A_CF V_FF q_F
7:      p_C ← BIF(k − 1, q_C)
8:      p_F ← V_FF (q_F − A_FC p_C)
9:      p ← (p_F, p_C)
10:   end if
11: return p .
```
(4.4)

Here (like in the following presentation) the lower index C for the coefficient matrix on the coarse level is dropped. The operation $(q_F, q_C) \leftarrow q$ in step 5, which separates a vector $q \in \mathsf{R}^n$ into the components $q_F \in \mathsf{R}^{(n-n^C)}$ belonging to the set of removed unknowns and components $q_C \in \mathsf{R}^{n^C}$ belonging to the set of coarse level unknowns, is called restriction. The inverse operation $p \leftarrow (p_F, p_C)$ in step 9 is called prolongation. The forward substitution in step 6 and backward substitution in step 8 need matrix-vector multiplications with the matrices A_{CF} and A_{FC}. Moreover matrix-vector multiplications with the coefficient matrix on the coarsest level l are needed in order to perform the iterative solver. As we want to get a fast evaluation of M^{-1} these matrices may have only a view diagonals with non–zero entries in order to store these matrices efficiently in the diagonal storage scheme. This requirement has to be considered when selecting the coarse level unknowns C.

5. AMLI

In order to get a stable block incomplete factorisation we assume that the matrix A is an M–matrix, see [2]. Then there is a vector $v^{pos} \in \mathsf{R}^n$ with

$$v^{pos} > 0 \text{ and } Av^{pos} > 0 \tag{5.1}$$

($v > 0$ iff for all $i = 1, \ldots, n$ $v_i > 0$).

The positive vector v^{pos} is used to construct the diagonal matrix V_{FF} of the matrix A_{FF} in the approximate factorisation (4.3). V_{FF} is calculated from the equation

$$V_{FF} A_{FF} v_F^{pos} = v_F^{pos} . \tag{5.2}$$

The entries of the main diagonal in V_{FF} are positive due to the M–matrix properties of A and the definition of v^{pos}. Moreover it can be shown that the new coarse level matrix fulfils the condition (5.1) with the positive vector v_C^{pos} and for all levels V_{FF} has positive main diagonal entries (ie. there is no break down) independent from the selection of the coarse grid.

In the case that A is symmetric the coarse level matrices are positive definite which is important for a fast iterative solution of the linear system with a CG method. For this case a very well–developed analysis is available, see [1]. The corresponding preconditioned CG method is called algebraic multilevel iteration (AMLI). However, the analysis was given for a W–cycle type implementation, but by solving a sufficiently large system at the lowest level l the computing time for a V-cycle method (4.4) is optimal, see [5].

6. Construction of v^{pos}

Unfortunately there is no algorithm available to calculate v^{pos} which is significantly cheaper than the costs for the solution of the original system (1.1). Therefore a heuristic method, which is not always but mostly successful, is used to construct a suitable v^{pos} cheaply.

Motivated by the assumption that the matrix is produced by the discretisation of a boundary value problem it is assumed the v^{pos} is a d-dimensional polynomial

$$v^{pos} := \sum_{p=0}^{p_0} \alpha_p \lambda^p \tag{6.1}$$

where it is

$$\lambda^p = (\prod_{i=1}^{d} (\frac{s_i}{n_i})^p)_{s_1=1,\ldots,m_1;\ldots;s_d=1\ldots,m_d} \cdot \tag{6.2}$$

In the case that $A\lambda^0 > 0$ is not fulfilled (in which case it is $p_0 = 0$) the values for $(\alpha_p)_{p=0,\ldots,p_0}$ are calculated successively in the following way:

Starting with $p = 1$ and $v = 1$ we try to find β with

$$\begin{aligned} v + \beta\lambda^p &> 0 \\ Av + \beta A\lambda^p &> 0 \,. \end{aligned} \tag{6.3}$$

Using the fact that $\lambda_i^p > 0$ for all $i = 1, \ldots, n$ the following conditions for β can be derived:

$$\begin{aligned} \beta &> \beta_0 := \min_i \frac{\lambda_i^p}{v_i} \\ \beta &> \beta_{min} := -\min_{(A\lambda^p)_i>0} \frac{(Av)_i}{(A\lambda^p)_i} \\ \beta &< \beta_{max} := -\max_{(A\lambda^p)_i<0} \frac{(Av)_i}{(A\lambda^p)_i} \end{aligned} \tag{6.4}$$

here we set $\max \emptyset = -\infty$ and $\min \emptyset = \infty$.

In the case that $\max(\beta_0, \beta_{min}) < \beta_{max}$ we can set $\beta = \frac{1}{2}(\max(\beta_0, \beta_{min}) + \beta_{max})$ (and $\beta = |\max(\beta_0, \beta_{min})| + 1$ if $\beta_{max} = \infty$). The vector $v^{pos} = v + \beta\lambda^p$ fulfils the positivity conditions (5.1).

If $\max(\beta_0, \beta_{min}) \geq \beta_{max}$ (ie. there is no β to fulfill condition (6.3) we set $\beta = 1.05 \max(\beta_0, \beta_{min})$ if $\max(\beta_0, \beta_{min}) > 0$ and we set $\beta = 0.95 \max(\beta_0, \beta_{min})$ if $\max(\beta_0, \beta_{min}) > 0$. Using this setting we ensure that the first condition in (6.3) is fulfilled and becomes likely that the second condition hold for a large number components. With $v := v + \beta\lambda^p$ and $p := p + 1$ a new β is calculated to fulfill conditions (6.3).

We found that this approach works very good in many applications. However, the algorithm can fail even if A is an M–matrix. However, if the algorithm is not successful after $p_0 = 4$ steps it is set $v^{pos} = (1, \ldots, 1) = \lambda^0$. By experience we learned that if the main diagonal of V_{FF} contains only positive entries AMLI works still very well, although v^{pos} does not have the required properties. Therefore the factorisation is stopped if the matrix condensation (5.2) produces a non–positive diagonal matrix.

7. Definition of Coarse Level

The selection of the coarse level unknowns C bases on an alternating direction approach: The coarse grid is constructed by removing every second grid hyperplane in a fixed spatial direction. The spatial direction is changed from coarsening step to coarsening step in order to achieve a uniform coarsening. The number of unknowns is approximatively halved in every coarsening step.

For instance starting from a $m_1 \times m_2$, 2-dimensional grid the first level is created by removing every second grid line in the x_1-direction The second level is created by removing every second grid line in the x_2–direction, see Figure 3. So the first level deals with $\frac{m_1+1}{2} \times m_2$ and the second level with $\frac{m_1+1}{2} \times \frac{m_2+1}{2}$ unknowns.

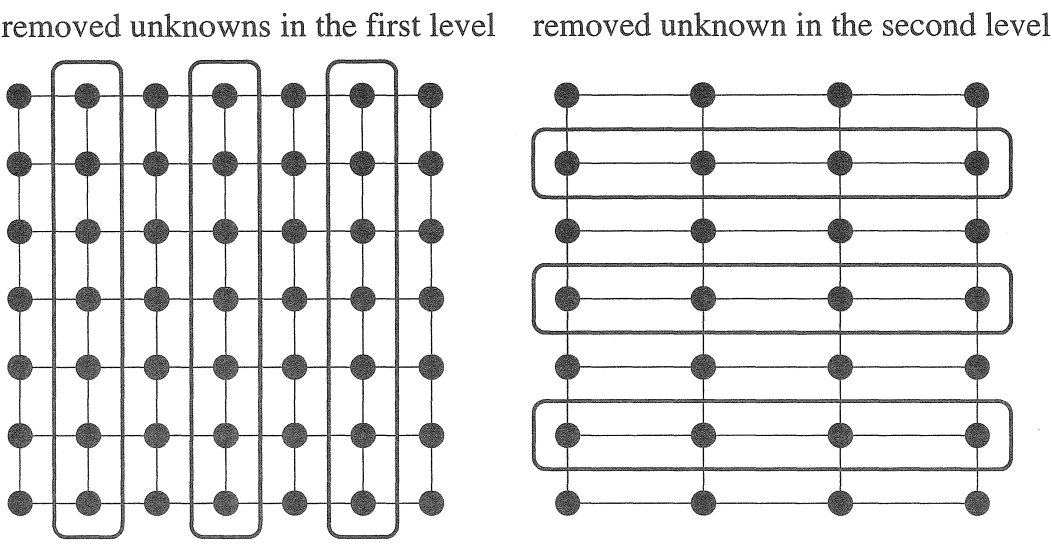

FIGURE 3. Definition of the two coarse level grids for a 7×7 grid.

7.1. Direction of Coarsening It turned out that in some applications it is not optimal to follow strictly the alternating direction approach, eg. if the number of grid points m_1 in the first grid direction is much larger than the number of grid points m_2 in the second grid direction. In this case it is better to execute two successive coarsening steps in the first grid direction before one in the second grid direction. The following selection strategy for the direction of coarsening is used:

It can be expected that the error of the condensation of the matrix block A_{FF} to the diagonal matrix V_{FF} is small if the entries of A_{FF} which are outside the main diagonal are small. Thus the block matrices A_{CF} and A_{FC} should contain the larger entries of the matrix. The entries in these matrix blocks belong to the diagonals $\pm D_i$ of A which represent the direction of coarsening according to definition (2.3).

So the grid is refined in that direction i where the sum of the corresponding diagonal entries

$$\sum_{k=1}^{n-D_i} |a_{k,k+D_i}| + \sum_{k=1+D_i}^{n} |a_{k,k-D_i}| \tag{7.1}$$

is maximal. As the size of the entries in the diagonal D_i are proportional to the square of the number of grid points m_i in the corresponding spatial direction, condition (7.1) picks up this direction for coarsening (if the underlying boundary value problem is isotropic).

7.2. Numbering of removed unknowns Usage of the lexicographic ordering for the grid point of the coarse level and the set of removed unknowns creates matrices A_{CC} and A_{FF} which both contain only a small number of diagonals with non–zero entries. When using a $2d+1$-point stencil for the discretisation the number of diagonals is $2d-1$ for both matrices.

Unfortunately this is not necessarily true for the matrices A_{FC} and A_{CF}, namely if the number of grid points in the direction of coarsening is odd. Figure 4 shows the example of a 5×5 grid with coarsening in the first spatial direction. The number on the connection

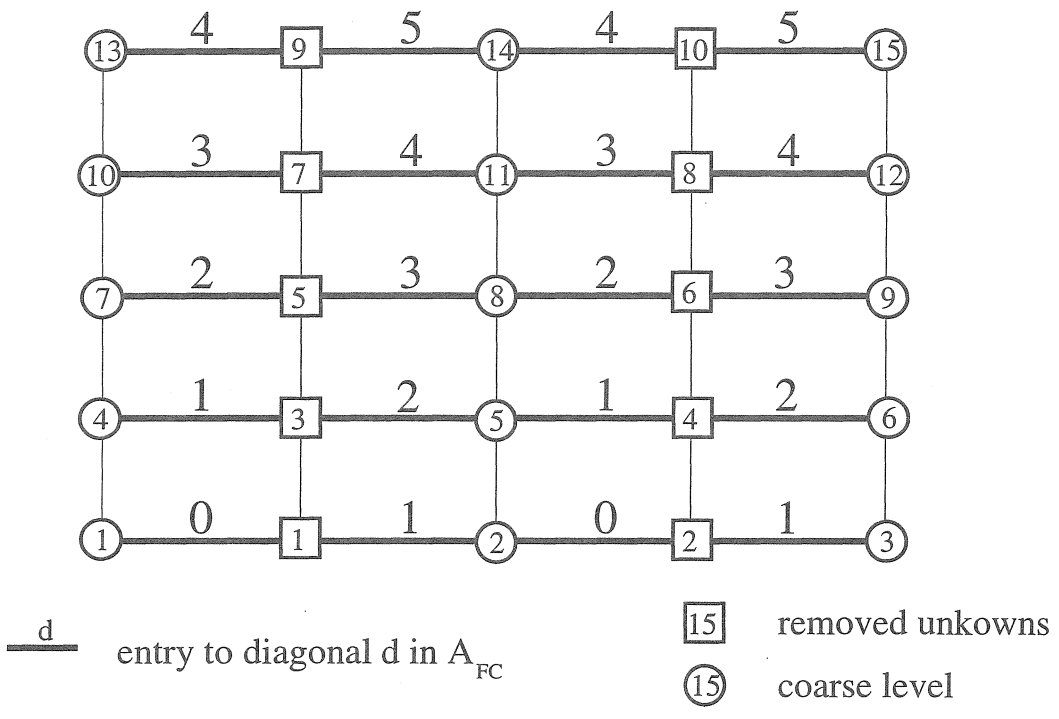

FIGURE 4. Canonical numbering of removed unknowns.

between a coarse level and a removed node shows the diagonal of the corresponding entry in A_{FC}. Obviously in this example the number of diagonals with non–zero entries is 6 (in case of an $m_1 \times m_2$-grid it is $m_2 + 1$). In a general situation the number of diagonals is too large to achieve a fast matrix-vector multiplication using a diagonal storage scheme.

Using the observation that this situation does not occur if the number of grid point in the direction of coarsening is even a dummy grid point is introduced. The grid points on the created hyperplane which is orthogonal to the direction of coarsening are added to the set of removed unknowns F. The corresponding additional rows in A_{FC} and columns in A_{CF} are set to zero. As now the grids of coarse level unknowns and removed unknowns are interlocked like a zipper the number of diagonals with non–zero entries in A_{FC} and A_{CF} is minimal, see Figure 5. In the case of a $2d+1$-point stencil the number of diagonals is two for both matrices.

8. Assemblage of the Schur Complement

As we have seen in the previous section all sub-matrices contains only a few diagonals with non–zero entries. We will see that this is also true for the approximate Schur complement A^C.

If Δ^C denotes the set of diagonals of the Schur complement A^C and $d \in \Delta^C$ we get for the entries in diagonal d by equation (4.2)

$$(a^C)_{i,i+d} = (a_{CC})_{i,i+d} - \sum_{k=1}^{n^C} (a_{CF})_{i,k} (\hat{a}_{FC})_{k,i+d} \tag{8.1}$$

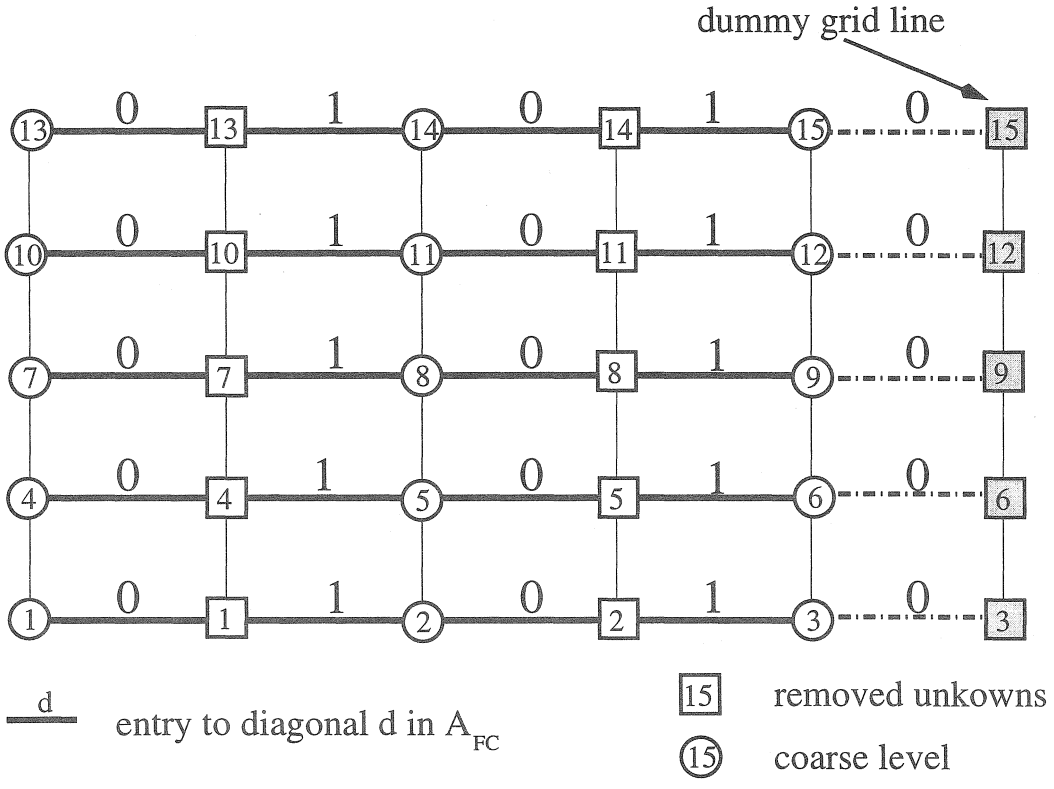

FIGURE 5. Numbering of removed unknowns with dummy grid line.

for all $i = 1, \ldots, n^C$, where $\hat{A}_{FC} := V_{FF} A_{FC}$. Using the fact that A_{CF} is only non-zero for entries in the diagonals Δ_{CF} we get

$$(a^C)_{i,i+d} = (a_{CC})_{i,i+d} - \sum_{e \in \Delta_{CF}} (a_{CF})_{i,i+e} (\hat{a}_{FC})_{i+e,i+d} \tag{8.2}$$

for all $i = 1, \ldots, n^C$. Consequently the diagonal d of the approximate Schur complement A^C can only contain non–zero entries if d is a diagonal with non–zero entries of A_{CC} (ie. $d \in \Delta_{CC}$) or there is a diagonal $e \in \Delta_{CF}$ in such a way that $d - e$ is a diagonal with non–zero entries of A_{FC} (ie. $d - e \in \Delta_{FC}$):

$$\Delta^C \subset \Delta_{CC} \cup (\Delta_{CF} + \Delta_{FC}) \,. \tag{8.3}$$

This shows that the number of diagonals with non–zero entries in the approximate Schur complement A^C is small as Δ_{CC}, Δ_{CF} and Δ_{FC} contain only a small number of elements. Actually in the case of a $2d+1$-point stencil and coarsening in the first spatial direction it is $\Delta_{CF} = \{-1, 0\}$, $\Delta_{FC} = \{0, -1\}$ and $\Delta_{CC} = \{\pm D_i | i = 2, \ldots, d\} \cup \{0\}$. From formula (8.3) we get $\Delta^C \subset \Delta$ (as $D_0 = 1$).

In a general situation formula (8.3) is evaluated to get an estimate for the number of diagonals with non–zero entries in A^C. This allows to provide the storage needed for the diagonals of the approximate Schur complement in order to assemble this matrix. The

diagonals of A_{CC} are copied into this array and the remaining diagonals are initialised with zeros. To evaluate the sum in equation (8.2) we take a diagonal $e \in \Delta_{CF}$ of A_{CF} and a diagonal $\hat{e} \in \Delta_{FC}$ of $\hat{A}_{FC}$ and multiply both component-by-component, where in the diagonal $\hat{e}$ on offset of e is used. The result vector is subtracted from diagonal $e + \hat{e}$ of the approximate Schur complement. Notice that these assemblage technique requires $\kappa(\Delta_{CF}) \cdot \kappa(\Delta_{FC})$ vector operations with vector length n^C (actually it is $n - n^C$ but because of the dummy grid points it is n^C). After all diagonals of A_{CF} and $\hat{A}_{FC}$ have been processed diagonals containing no non-zero entries are removed from the array of the diagonals of the Schur complement.

It is emphasised that the small number of diagonals with non–zero entries in A_{FC} and A_{CF} not only provides an efficient matrix–vector multiplication in the forward- and backward substitution but also a very fast diagonal-by-diagonal scheme for the assemblage of the approximate Schur complement. The latter ensures that the start–up phase to calculate approximate factorisations does not devour the acceleration of the iterative solver by the preconditioner.

9. Optimal Number of Levels

The selected number of levels has to be large enough to ensure that the coarse level system is small and can be solved very fast. On the other hand a large number of levels produces a significant loss of information during the coarsening procedure (which have to be compensated by a W-cycle method). So the optimal number of levels has to balance both effects.

If the number of operations per unknown for the forward/backward substitution is equal to C_0 and $n^{(k)}$ denotes the number of unknowns in level k the total complexity for the forward/backward substitution $C_{fb}(l)$ is

$$C_{fb}(l) = \sum_{k=1}^{l} C_0 n^{(k)} \approx \sum_{k=1}^{l} C_0 \frac{n}{2^k} \approx C_0 n \tag{9.1}$$

where we use the fact that the number of grid points is nearly halved in every coarsening step and assume that the l is sufficiently large.

When using a CG method for the solution of the coarse level system the complexity per iteration step is proportional to $n^{(l)}$. The number of iteration steps, which are needed to reach a certain accuracy, is assumed to be proportional to the number of matrix-vector multiplications that are needed to transfer a perturbation from one end of the grid to the other (a very optimistic assumption!). Thus, the complexity for the solution of the coarse level system is

$$C_{ls}(l) = C_1 n^{(l)} \max_{i=1,\ldots,d} m_i^{(l)} \tag{9.2}$$

with a constant $C_1 > 0$.

A large number of levels l would make $C_{ls}(l)$ very small and would so minimise the total complexity $C_{fb}(l) + C_{ls}(l)$ for the evaluation of M^{-1}. On the other hand l has to be small in order to limit the number of outer iteration steps. If $C_{ls}(l)$ and $C_{fb}(l)$ are equal, ie. the same computational effort is invested to the forward/backward substitution as well as to the solution of the coarse level problem, we can expect to achieve optimal

computing time. So the coarsening procedure is stopped after level l if the condition

$$n^{(l)} \max_{i=1,\ldots,d^{(l)}} m_i^{(l)} \leq n \tag{9.3}$$

is fulfilled (where we assume that $C_0 \approx C_1$).

By setting $m_i^{(l)} \approx (n^{(l)})^{\frac{1}{d}}$ for all $i = 1, \ldots, d$ and $n^{(l)} = \frac{n}{2^l}$ condition (9.3) is simplified to

$$l = \frac{1}{d+1} \log_2 n \tag{9.4}$$

(see [6]). Thus a typical number of optimal levels is in the order six. Notice that the number of optimal levels grows very slowly with the number of unknowns n. In case of a 3-dimensional grid the number of grid points has to be increased by a factor of 16 before an additional level should be used. This indicates that the valley of optimal values for l is rather flat. This is likely the reason why the condition (9.3) delivers very good results (see next section) although its derivation seems to be very far–fetched.

10. Examples

In the following examples the timings of the presented AMLI implementation is measured on the Fujitsu VPP300 (with peak performance 2.2Gflops). The coefficient matrices A are generated by the discretisation of boundary value problems and normalised symmetrically to make the main diagonal entries equal to one. The right hand side b of the linear system (1.1) is calculated from a solution u which is generated by a random sample. The linear systems are solved with an accuracy $TOL = 10^{-6}$. For the inner iteration the accuracy 10^{-3} is used. In cases of a symmetrical coefficient matrix ORTHOMIN is used on the finest and coarsest level. Otherwise GMRES with truncation after five residuals and restart after 20 iteration step is used on the finest and GMRES with truncation after ten residuals and without restart is used on the coarsest level.

All timings are given in seconds and consider the whole solution procedure including the assemblage of the coarse level matrices. If the timings are given in the form *'time/perf'* *time* is the computing time in seconds and *perf* the performance in million floating point operations per second (*Mflops*). The performance considers only the iteration procedure but does not include the assemblage of the coarse level matrices. Columns entitled with 'Jacobi' refer to tests with Jacobi preconditioning but without multilevel preconditioning (ie. $l = 0$).

10.1. 2D Poisson Equation The first test case is the 2-dimensional Poisson equation

$$\begin{aligned} -\nabla \cdot \nabla u &= f \quad \text{on} \quad \Omega \\ u &= \phi \quad \text{on} \quad \partial\Omega \end{aligned} \tag{10.1}$$

on the unit square $\Omega = [0,1]^2$. The equation is discretised by the finite difference method using a rectangular $\sqrt{n} \times \sqrt{n}$ grid.

The timings on one compute node for several grids and several number of levels is presented in Table 1. The smallest CPU times for a particular problem size is marked by using the **bold** font. Obviously the condition (9.3) for the selection of the number of levels picks the optimal number of levels. However, this is not too difficult as the valley of nearly optimal numbers of levels is rather flat.

l	n		
	123585	492929	1968897
0	4.13/785	17.50/856	233./839
1	8.23/793	59.60/829	448./837
2	3.49/704	22.40/806	157./813
3	2.28/637	12.20/770	80.6/791
4	1.31/504	6.02/688	30.1/804
5	1.12/393	4.27/626	20.4/753
6	**0.92/320**	3.02/538	12.7/683
7	1.03/293	**3.09/487**	11.3/611
8	0.93/297	2.74/472	**9.95/571**
9	1.11/308	3.29/471	11.7/563
10	1.08/317	3.20/483	11.5/555
11	1.21/323	3.85/497	14.3/558

TABLE 1. Computing time for 2D Poisson equation

	$R_y = 1$		$R_y = 10$		$R_y = 100$	
n	AMLI	Jacobi	AMLI	Jacobi	AMLI	Jacobi
274625	2.57/662	7.32/1012	3.48/639	17.6/908	16.9/669	83.9/907
571787	5.11/737	19.3/1036	6.95/768	49.8/944	32.3/801	180./944
1092727	10.8/677	45.9/1040	14.7/790	134./935	60.1/862	>900
1685159	16.2/801	80.6/1049	24.5/841	249./961	90.6/874	>900

TABLE 2. Convection driven diffusion (one compute node)

		$R_y = 1$		$R_y = 10$		$R_y = 100$	
n	p	AMLI	Jacobi	AMLI	Jacobi	AMLI	Jacobi
571787	1	5.11/737	19.3/1036	6.95/768	49.8/944	32.3/801	180./944
1092727	2	6.39/1147	23.8/1996	10.1/1151	99.4/1835	46.0/1053	569./1886
1685159	4	6.04/1952	21.0/4033	8.92/2026	83.7/3957	40.2/2035	513./3964
3176523	8	6.34/3862	25.6/7640	11.1/3816	104./7496	52.9/3729	685./7490

TABLE 3. Convection driven diffusion

The computing times for the optimal number of levels increase less then linearly with the number of unknowns, although the complexity grows slightly faster than linearly (as a V-cycle is used). But the performance for the larger problems is much better since the coarse level system is larger and thus the solver works with longer vectors. Notice that a smaller number of levels improves the performance but does not reduce the computing time.

10.2. Convection driven diffusion The next test case is the 3-dimensional equation for convection driven diffusion

$$\begin{aligned} -\nabla \cdot \nabla u + b\nabla u &= f \quad \text{on} \quad \Omega \\ u &= \phi \quad \text{on} \quad \partial\Omega \end{aligned} \tag{10.2}$$

on the domain $[-1,1]^3$. The vector-valued coefficient function b which represents a velocity field b is defined by

$$b = \left(\frac{\partial\psi}{\partial x_2} - \frac{\partial\psi}{\partial x_3}, \frac{\partial\psi}{\partial x_3} - \frac{\partial\psi}{\partial x_1}, \frac{\partial\psi}{\partial x_1} - \frac{\partial\psi}{\partial x_2}\right) \tag{10.3}$$

with stream function ψ defined by

$$\psi = R_y(x_1^2 - 1)(x_2^2 - 1)(x_3^2 - 1)\sqrt{x_1^2 + x_2^2 + x_3^2}\,. \tag{10.4}$$

The equation is discretized by the finite difference method using a rectangular $n^{\frac{1}{3}} \times n^{\frac{1}{3}} \times n^{\frac{1}{3}}$ grid where an up-wind scheme is used for the convective term $b\nabla$. The problem is a hard case for iterative solvers as for large values of R_y the coefficient matrix is singular.

Table 2 presents the timings and performances for a series of grids on one compute node. The computing time for AMLI grows linearly with the number of unknowns independent from the value for R_y which controls the influence of the convection term. The performance is rather high compared to the performance for the iteration without preconditioning. This arises from the fact that the solution of the coarse level system is rather expensive and thus the solver spends most of the time with solving the coarse level system where a very good performance can be achieved.

This changes when more then one compute node is used, see Table 3. The calculation achieves a quite good scalability (at least for a small value of R_y) but the performance is rather poor compared to the performance that is reached without preconditioning. The gather and scatter operations over the compute nodes needed for the restriction and prolongation slows down the calculation. Another factor is the low performance per compute node during the parallel solution of the coarse level system because the vector length per compute node is rather small as well as the part of time spent for communication is larger. To solve the coarse level system on only one compute node does not speed-up the calculation as the available computing power is cut down.

Although the performance is not optimal, AMLI reduces the computing time dramatically on one as well as on several compute nodes.

10.3. Finite Element Problem In order to demonstrate the flexibility of the code it is applied to the finite element discretisation of the 2-dimensional variational problem

$$\begin{aligned} \int_\Omega (k\nabla v \cdot \nabla u - vf)\, d\Omega = 0 \quad &\text{for all smooth} \quad v : \Omega \to \mathsf{R} \\ u = \phi \quad &\text{on} \quad \partial\Omega\,. \end{aligned} \tag{10.5}$$

on the domain $[0,1]^2$. The problem is discretized by using finite elements of order one on a triangulation of $[0,1]^2$ that bases on a rectangular grid. The coefficient function k is selected by assigning every element a random value in the interval $[1, 1000]$. The matrices were generated by the finite element code VECFEM, see [3]. Table 4 shows the reduction of the calculation time by at least a factor of 5. The heavy oscillations of the timings are effected by the random sample of the values for k.

n	l	AMLI	Jacobi
12544	5	0.26	0.19
25600	5	0.39	0.51
50176	5	0.65	1.20
99856	6	0.86	3.05
202500	6	1.34	7.75
403225	7	4.16	23.9

TABLE 4. Finite element problem

11. Conclusion

The assumption that the coefficient matrix A is an M–matrix allows to select the coarse level nodes in a way which is optimal for the used computer architecture. In (nearly) any case the construction procedure does not break down and the constructed preconditioner is successful in reducing the number of iteration steps. The rectangular grid provides the possibility to get optimal data flow for SIMD architectures.

However, both assumptions are very restrictive and do not provide the functionality required by users. When dropping the M–matrix properties a more careful selection of the coarse level unknowns has to be applied as the actual entries in the matrix have to be considered. Thus the data structures have to consider unstructured grids. Unfortunately the introduction of suitable data structures will reduce the achievable performance as indexed memory accesses are needed.

References

[1] Axelsson, O., and Neytcheva, M. Algebraic Multilevel Iteration Method for Stieltjes Matrices. *Num. Lin. Alg. Appl. 1*, 1994, 213–236.

[2] Berman, A., and Plemmons, R. J. *Nonnegative Matrices in Mathematical Science.* SIAM, Philadelphia, US, 1994.

[3] Grosz, L., Roll, C., and Schönauer, W. VECFEM for mixed Finite Elements. Internal report 50/93, University of Karlsruhe, Computing Center, Postfach 6980, 76128 Karlsruhe, Germany, 1993.

[4] Hackbusch, W. *Multigrid Methods and Applications.* Springer-Verlag, Berlin, Heidelberg, New York, 1985.

[5] Neytcheva, M. Experience in Implementing the Algebraic Multilevel Iteration Method on a SIMD–type computer. *Appl. Numer. Math. 19*, 1995, 71–90.

[6] Neytcheva, M., Adiy, A., Mellaard, M., Geogiev, K., and Axelsson, O. Scalable and Optimal Iterative Solvers for Linear and Nonlinear Problems. Final Report 9613, Department of Mathematics, University of Nijmegen, The Netherlands, 1996.

[7] Saad, Y. ILUM: A Multi-Elimination ILU Preconditioner for General Sparse Matrices. *SIAM J. Sci. Comput. 17*, 1996, 830–847.

[8] Schönauer, W. *Scientific Computing on Vector Computers.* North-Holland, Amsterdam, New York, Oxford, Tokyo, 1987.

[9] Weiss, R. *Parameter-Free Iterative Linear Solvers.* Mathematical Research, vol. 97. Akademie Verlag, Berlin, 1996.