GPU Implementation of the RRB-solver

de Jong, Martijn; Vuik, Kees

Publication date
2016

Document Version
Final published version

Citation (APA)
de Jong, M., & Vuik, K. (2016). *GPU Implementation of the RRB-solver*. (Reports of the Delft Institute of Applied Mathematics; Vol. 16-06). Delft University of Technology.

DELFT UNIVERSITY OF TECHNOLOGY

REPORT 16-06

GPU implementation of the RRB-solver

M.A. de Jong, and C. Vuik

ISSN 1389-6520

Reports of the Delft Institute of Applied Mathematics

Delft 2016

1 Introduction

Throughout this report a linear system of equations

$$A\mathbf{x} = \mathbf{b} \tag{1.1}$$

is considered, where $A \in \mathbb{R}^{n \times n}$ is a large symmetric positive definite (SPD) pentadiagonal matrix, $\mathbf{x} \in \mathbb{R}^n$ the solution vector, and $\mathbf{b} \in \mathbb{R}^n$ the right-hand side (RHS) vector. Among other solution methods, see Figure 1, the Conjugate Gradient (CG) algorithm, an example of a Krylov method, is a good choice to solve (1.1) with the listed properties.

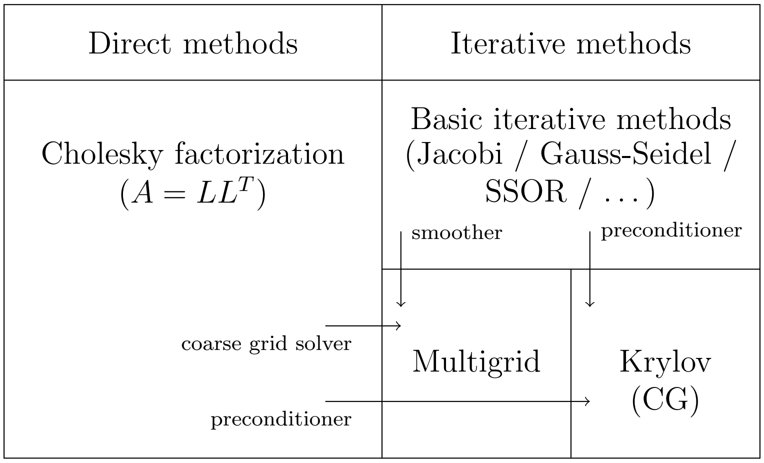


Figure 1: Solution methods to solve (1.1)*. Methods based on (incomplete) Cholesky factorization and basic iterative methods (BIMs) can be used to construct a preconditioner for CG.*

In exact arithmetic, the CG algorithm finds an approximate solution such that the A-norm of the residual is minimized over the Krylov subspace

$$\mathcal{K}_k(A, \mathbf{r}_0) = \operatorname{span}\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0\},$$

where $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ is the initial residual. A frequently used upper bound for the error after k steps is [9]:

$$\|\mathbf{x}_k - \mathbf{x}\|_A \leq 2\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}\right)^k \|\mathbf{x}_0 - \mathbf{x}\|_A, \tag{1.2}$$

where $\|\cdot\|_A$ denotes the A-norm, and $\kappa(A)$ the spectral condition number of matrix A defined as follows:

$$\kappa(A) := \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)},$$

where $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ are the largest and smallest eigenvalue of A, respectively. We see that, in general, the smaller $\kappa(A)$, the better the convergence rate. The convergence rate can often strongly be improved by applying CG to a preconditioned system

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}, \tag{1.3}$$

where non-singular matrix M is called the preconditioner. The matrix M is constructed such that $\kappa(M^{-1}A) \ll \kappa(A)$.

When the CG algorithm is applied to the preconditioned system (1.3) the preconditioned conjugate gradient (PCG) algorithm is obtained, see Algorithm 1 (cf. [8]; Algorithm 9.1).

Algorithm 1 The PCG algorithm.

```
 1.  $\mathbf{r} = \mathbf{b} - A\mathbf{x}$
 2.  Solve $M\mathbf{z} = \mathbf{r}$ for $\mathbf{z}$
 3.  $\rho_1 = \langle \mathbf{r}, \mathbf{z} \rangle$
 4.  Set $\mathbf{p} = \mathbf{z}$
 5.  While (not converged)    % Termination criterium
 6.    $\rho_0 = \rho_1$
 7.    $\mathbf{q} = A\mathbf{p}$                  % Matrix-vector product
 8.    $\sigma = \langle \mathbf{p}, \mathbf{q} \rangle$             % Inner product
 9.    $\alpha = \rho_0 / \sigma$
10.    $\mathbf{x} = \mathbf{x} + \alpha\mathbf{p}$              % Vector update
11.    $\mathbf{r} = \mathbf{r} - \alpha\mathbf{q}$              % Vector update
12.    Solve $M\mathbf{z} = \mathbf{r}$           % Preconditioner step
13.    $\rho_1 = \langle \mathbf{r}, \mathbf{z} \rangle$             % Inner product
14.    $\beta = \rho_1 / \rho_0$
15.    $\mathbf{p} = \mathbf{z} + \beta\mathbf{p}$              % Vector update
16.  End while
```

Lines 5 to 15 are repeated until the predefined termination criterium is met at Line 5. Per iteration of PCG we have 3 vector updates, 2 inner products, a matrix-vector product and a preconditioner step

$$\text{Solve } M\mathbf{z} = \mathbf{r} \text{ for } \mathbf{z}.$$

This step should be relatively cheap; the computational costs should be not much larger than those for computing the matrix-vector product with A.

2 The RRB-solver

In this report we focus on the so-called RRB-solver to solve (1.1). The RRB-solver is a PCG-type solver where the RRB-factorization algorithm [1, 2, 3] is used to construct the preconditioner M. RRB stands for “Repeated Red-Black” (or “Recursive Red-Black”) which refers to how nodes in a two dimensional grid are colored and numbered.

2.1 RRB-numbering

Let

$$G = \left\{ (i, j) \mid 1 \le i \le N_x, 1 \le j \le N_y \right\}$$

be the set of all nodes in an $N_x \times N_y$ grid. If $(1,1)$ is chosen to be a black node, then a standard red-black ordering is given by:

$$\begin{aligned} R^{[1]} =& \big\{(i,j) \in G \mid \mathrm{mod}(i+j,2)=1\big\}, \\ B^{[1]} =& \; G \setminus R^{[1]}, \end{aligned}$$

where $R^{[1]}$ denotes the set of first level red nodes and $B^{[1]}$ denotes the set of first level black nodes. Next, a standard red-black ordering is applied a second time to the $B^{[1]}$-nodes as follows:

$$\begin{aligned} R^{[2]} =& \big\{(i,j) \in B^{[1]} \mid \mathrm{mod}(j,2)=0\big\}, \\ B^{[2]} =& \; B^{[1]} \setminus R^{[2]} \\ =& \; G \setminus (R^{[1]} \cup R^{[2]}). \end{aligned}$$

The second level black nodes, i.e., the $B^{[2]}$-nodes, are thus the nodes in G that neither belong to the sets $R^{[1]}$ nor $R^{[2]}$. Generally,

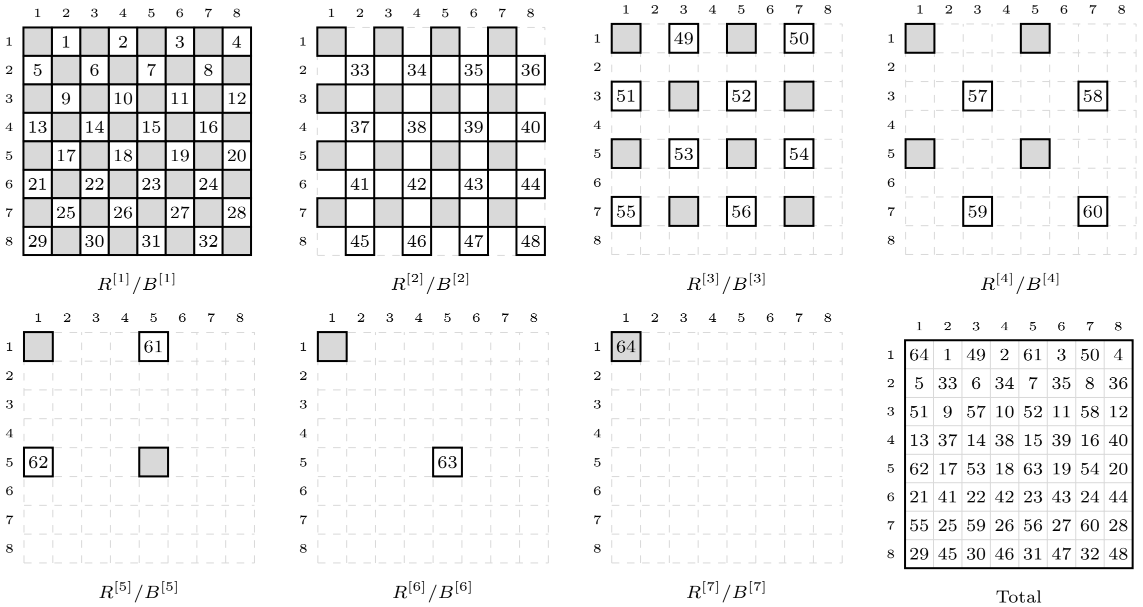
$$R^{[k]} = \begin{cases} \Big\{(i,j) \in G \setminus \big(\cup_{p=1}^{k-1} R^{[p]}\big) \mid \mathrm{mod}\big(i+j, 2^{\frac{k+1}{2}}\big) = 2^{\frac{k-1}{2}}\Big\}, & k \text{ odd;} \\ \Big\{(i,j) \in G \setminus \big(\cup_{p=1}^{k-1} R^{[p]}\big) \mid \mathrm{mod}\big(j, 2^{\frac{k}{2}}\big) = 0\Big\}, & k \text{ even.} \end{cases}$$

The maximum number of levels that the $N_x \times N_y$-grid allows for is given by

$$\ell_{\max} = 2\lceil \log_2(\max\{N_x, N_y\})\rceil + 1. \tag{2.1}$$

Example

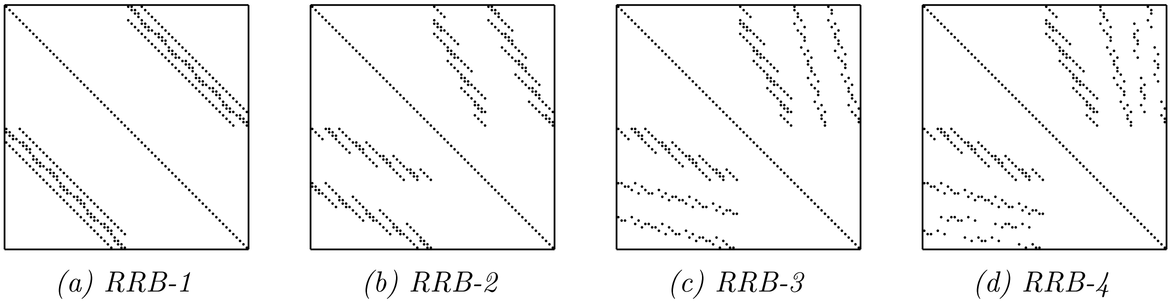
In this example the RRB-numbering procedure is applied to a matrix $A \in \mathbb{R}^{64\times 64}$ resulting from an 8×8 grid of unknowns. For this matrix A the maximum number of levels is $\ell_{\max} = 2\lceil \log_2(8)\rceil + 1 = 7$ according to Equation (2.1). The result of the RRB-numbering on the ordering can be seen in Figure 2. For readability the black nodes are represented by gray squares and the red nodes by white squares.



	1	2	3	4	5	6	7	8
1	64	1	49	2	61	3	50	4
2	5	33	6	34	7	35	8	36
3	51	9	57	10	52	11	58	12
4	13	37	14	38	15	39	16	40
5	62	17	53	18	63	19	54	20
6	21	41	22	42	23	43	24	44
7	55	25	59	26	56	27	60	28
8	29	45	30	46	31	47	32	48

Figure 2: RRB-numbering for an 8×8 *grid.*

The effect of the RRB-numbering on the sparsity pattern of the matrix $A \in \mathbb{R}^{64 \times 64}$ belonging to the 8×8 grid is shown in Figure 3 for the first four levels.



(a) RRB-1 *(b) RRB-2* *(c) RRB-3* *(d) RRB-4*

Figure 3: Effect of RRB-numbering on sparsity pattern.

2.2 The RRB-factorization algorithm

The RRB-factorization algorithm factorizes the matrix A into

$$A = LDL^T + R, \tag{2.2}$$

where L is a lower triangular matrix with unitary diagonal entries (1s), D a diagonal matrix, and R a matrix that contains adjustments resulting from lumping procedures. The RRB-factorization algorithm can be understood best by looking at two levels of red-black reordering, i.e., RRB-2, see Figure 4. Moreover, it is convenient to suppose, for now,

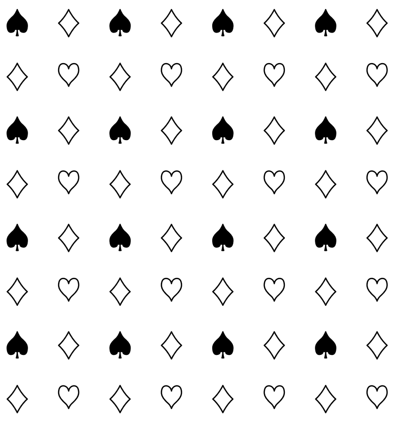


Figure 4: First level red nodes (diamonds) and second level red-black nodes (hearts and spades).

that A is initially given by a 9-point stencil rather than by a 5-point stencil, i.e., suppose that matrix A is given by the stencil

$$\begin{bmatrix} NW^{[0]} & N^{[0]} & NE^{[0]} \\ W^{[0]} & C^{[0]} & E^{[0]} \\ SW^{[0]} & S^{[0]} & SE^{[0]} \end{bmatrix} \quad \text{or, shortly denoted,} \quad \begin{bmatrix} NW & N & NE \\ W & C & E \\ SW & S & SE \end{bmatrix}^{[0]},$$

where the coefficient $N^{[0]}$ gives the coupling with the North neighbor, $W^{[0]}$ the West neighbor and so on. The superscript expresses the current level.

Using the RRB-2 ordering the linear system $A\mathbf{x} = \mathbf{b}$, Equation (1.1), can be written as

$$\begin{pmatrix} A_{\diamondsuit} & A_{\diamondsuit\heartsuit} & A_{\diamondsuit\spadesuit} \\ A_{\heartsuit\diamondsuit} & D_{\heartsuit} & A_{\heartsuit\spadesuit} \\ A_{\spadesuit\diamondsuit} & A_{\spadesuit\heartsuit} & D_{\spadesuit} \end{pmatrix} \begin{pmatrix} \mathbf{x}_{\diamondsuit} \\ \mathbf{x}_{\heartsuit} \\ \mathbf{x}_{\spadesuit} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_{\diamondsuit} \\ \mathbf{b}_{\heartsuit} \\ \mathbf{b}_{\spadesuit} \end{pmatrix},$$

where $D_{\heartsuit}$ and $D_{\spadesuit}$ are diagonal matrices, $A_{\diamondsuit}$, $A_{\heartsuit\spadesuit} = A^T_{\spadesuit\heartsuit}$ are four-banded matrices resulting from the NE-, SE-, SW- and NW-dependencies ($\times$) and $A_{\diamondsuit\heartsuit} = A^T_{\heartsuit\diamondsuit}$ and $A_{\diamondsuit\spadesuit} = A^T_{\spadesuit\diamondsuit}$ are four-banded matrices resulting from the N-, S-, W- and E-dependencies ($+$).

Stencil reduction (1st time)

For the $\diamondsuit$-nodes the 9-point stencil is reduced to a 5-point stencil ($+$) by using a so-called lumping procedure:

$$\begin{bmatrix} NW & N & NE \\ W & C & E \\ SW & S & SE \end{bmatrix}^{[0]} \Longrightarrow \begin{bmatrix} \cdot & N & \cdot \\ W & \widetilde{C} & E \\ \cdot & S & \cdot \end{bmatrix}^{[0]},$$

where $\widetilde{C}^{[0]} := C^{[0]} + NE^{[0]} + SE^{[0]} + SW^{[0]} + NW^{[0]}$, i.e., the four most outer elements are lumped towards the main diagonal. This means that

$$A = \widetilde{A}_0 + R_0, \tag{2.3}$$

i.e.,

$$A = \begin{pmatrix} \widetilde{D}_\diamondsuit & A_{\diamondsuit\heartsuit} & A_{\diamondsuit\spadesuit} \\ A_{\heartsuit\diamondsuit} & D_\heartsuit & A_{\heartsuit\spadesuit} \\ A_{\spadesuit\diamondsuit} & A_{\spadesuit\heartsuit} & D_\spadesuit \end{pmatrix} + \begin{pmatrix} R_\diamondsuit & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

where $\widetilde{D}_\diamondsuit$ is a diagonal matrix and where

$$R_\diamondsuit := A_\diamondsuit - \widetilde{D}_\diamondsuit$$

is the matrix of adjustments.

Elimination (1st time)

Elimination of the $\diamondsuit$-nodes by applying Gaussian elimination using $\widetilde{D}_\diamondsuit^{-1}$ yields

$$\widetilde{A}_0 = L_1 A_1 L_1^T, \tag{2.4}$$

where

$$L_1 = \begin{pmatrix} I_\diamondsuit & 0 & 0 \\ A_{\heartsuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1} & I_\heartsuit & 0 \\ A_{\spadesuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1} & 0 & I_\spadesuit \end{pmatrix}$$

and

$$A_1 = \begin{pmatrix} \widetilde{D}_\diamondsuit & 0 & 0 \\ 0 & D_\heartsuit - A_{\heartsuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1}A_{\diamondsuit\spadesuit} & A_{\heartsuit\spadesuit} - A_{\heartsuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1}A_{\diamondsuit\spadesuit} \\ 0 & D_{\spadesuit\heartsuit} - A_{\spadesuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1}A_{\diamondsuit\heartsuit} & D_\spadesuit - A_{\spadesuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1}A_{\diamondsuit\spadesuit} \end{pmatrix}.$$

The four lower-right blocks in A_1 correspond to a matrix given by a 9-point stencil:

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & N & \cdot & \cdot \\ \cdot & W & C & E & \cdot \\ \cdot & \cdot & S & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}^{[0]} - \frac{N^{[0]}}{C_N^{[0]}} \begin{bmatrix} \cdot & \cdot & N & \cdot & \cdot \\ \cdot & W & C & E & \cdot \\ \cdot & \cdot & S & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}_N^{[0]} - \frac{E^{[0]}}{C_E^{[0]}} \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & N & \cdot \\ \cdot & \cdot & W & C & E \\ \cdot & \cdot & \cdot & S & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}_E^{[0]}$$

$$- \frac{S^{[0]}}{C_S^{[0]}} \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & N & \cdot & \cdot \\ \cdot & W & C & E & \cdot \\ \cdot & \cdot & S & \cdot & \cdot \end{bmatrix}_S^{[0]} - \frac{W^{[0]}}{C_W^{[0]}} \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & N & \cdot & \cdot & \cdot \\ W & C & E & \cdot & \cdot \\ \cdot & S & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}_W^{[0]}$$

$$= \begin{bmatrix} \cdot & \cdot & N & \cdot & \cdot \\ \cdot & NW & \cdot & NE & \cdot \\ W & \cdot & C & \cdot & E \\ \cdot & SW & \cdot & SE & \cdot \\ \cdot & \cdot & S & \cdot & \cdot \end{bmatrix}^{[1]}.$$

Note: At this point one may stop and make a full Cholesky decomposition for the remainder given by the 9-point stencil.

Stencil reduction (2nd time)

For the $\heartsuit$-nodes the 9-point stencil is reduced to a 5-point stencil ($\times$) as well:

$$\begin{bmatrix} \cdot & \cdot & N & \cdot & \cdot \\ \cdot & NW & \cdot & NE & \cdot \\ W & \cdot & C & \cdot & E \\ \cdot & SW & \cdot & SE & \cdot \\ \cdot & \cdot & S & \cdot & \cdot \end{bmatrix}^{[1]} \Longrightarrow \begin{bmatrix} NW & \cdot & NE \\ \cdot & \widetilde{C} & \cdot \\ SW & \cdot & SE \end{bmatrix}^{[1]},$$

where $\widetilde{C}^{[1]} := C^{[1]} + N^{[1]} + E^{[1]} + S^{[1]} + W^{[1]}$. The result is

$$A_1 = \widetilde{A}_1 + R_1, \tag{2.5}$$

i.e.,

$$A_1 = \begin{pmatrix} \widetilde{D}_\diamondsuit & 0 & 0 \\ 0 & \widetilde{D}_\heartsuit & A_{\heartsuit\spadesuit} - A_{\heartsuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1}A_{\diamondsuit\spadesuit} \\ 0 & A_{\spadesuit\heartsuit} - A_{\spadesuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1}A_{\diamondsuit\heartsuit} & D_\spadesuit - A_{\spadesuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1}A_{\diamondsuit\spadesuit} \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & R_\heartsuit & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

with

$$R_\heartsuit := D_\heartsuit - A_{\heartsuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1}A_{\diamondsuit\heartsuit} - \widetilde{D}_\heartsuit.$$

Elimination (2nd time)

Elimination of the $\heartsuit$-nodes by applying Gaussian elimination using $\widetilde{D}_\heartsuit^{-1}$ yields

$$\widetilde{A}_1 = L_2 A_2 L_2^T, \tag{2.6}$$

where

$$L_2 = \begin{pmatrix} I_\diamondsuit & 0 & 0 \\ 0 & I_\heartsuit & 0 \\ 0 & (A_{\spadesuit\heartsuit} - A_{\spadesuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1}A_{\diamondsuit\heartsuit})\widetilde{D}_\heartsuit^{-1} & I_\spadesuit \end{pmatrix}$$

and

$$A_2 = \begin{pmatrix} D_\diamondsuit & 0 & 0 \\ 0 & \widetilde{D}_\heartsuit & 0 \\ 0 & 0 & \widehat{A}_\spadesuit \end{pmatrix}$$

with

$$\widehat{A}_\spadesuit := D_\spadesuit - A_{\spadesuit\diamondsuit}D_\diamondsuit^{-1}A_{\diamondsuit\spadesuit} - (A_{\spadesuit\heartsuit} - A_{\spadesuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1}A_{\diamondsuit\heartsuit})\widetilde{D}_\heartsuit^{-1}A_{\heartsuit\diamondsuit}D_\diamondsuit^{-1}A_{\diamondsuit\spadesuit}.$$

The matrix $\widehat{A}_\spadesuit$ is again given by a 9-point stencil but now on grid that is two times coarser in both x- and y-direction, i.e.,

$$\begin{bmatrix} \cdot & \cdot & N & \cdot & \cdot \\ \cdot & NW & \cdot & NE & \cdot \\ W & \cdot & C & \cdot & E \\ \cdot & SW & \cdot & SE & \cdot \\ \cdot & \cdot & S & \cdot & \cdot \end{bmatrix}^{[1]}$$

$$-\frac{NE^{[1]}}{C^{[1]}_{NE}}\begin{bmatrix} \cdot & \cdot & NW & \cdot & NE \\ \cdot & \cdot & \cdot & \widetilde{C} & \cdot \\ \cdot & \cdot & SW & \cdot & SE \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}^{[1]}_{NE} -\frac{SE^{[1]}}{C^{[1]}_{SE}}\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & NW & \cdot & NE \\ \cdot & \cdot & \cdot & \widetilde{C} & \cdot \\ \cdot & \cdot & SW & \cdot & SE \end{bmatrix}^{[1]}_{SE}$$

$$-\frac{SW^{[1]}}{C^{[1]}_{SW}}\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ NW & \cdot & NE & \cdot & \cdot \\ \cdot & \widetilde{C} & \cdot & \cdot & \cdot \\ SW & \cdot & SE & \cdot & \cdot \end{bmatrix}^{[1]}_{SW} -\frac{NW^{[1]}}{C^{[1]}_{NW}}\begin{bmatrix} NW & \cdot & NE & \cdot & \cdot \\ \cdot & \widetilde{C} & \cdot & \cdot & \cdot \\ SW & \cdot & SE & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}^{[1]}_{SW}$$

$$= \begin{bmatrix} NW & \cdot & N & \cdot & NE \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ W & \cdot & C & \cdot & E \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ SW & \cdot & S & \cdot & SE \end{bmatrix}^{[2]} .$$

Note: At this point one may stop and make a full Cholesky decomposition for the remainder given by the 9-point stencil.

RRB-iteration

The four steps listed above describe a first RRB-iteration. The first RRB-iteration thus consists of:

1. Stencil reduction for $\diamondsuit$-nodes (2.3): $A = \widetilde{A}_0 + R_0$.
2. Elimination of $\diamondsuit$-nodes (2.4): $\widetilde{A}_0 = L_1 A_1 L_1^T$;
3. Stencil reduction for $\heartsuit$-nodes (2.5): $A_1 = \widetilde{A}_1 + R_1$;
4. Elimination of $\heartsuit$-nodes (2.6): $\widetilde{A}_1 = L_2 A_2 L_2^T$;

All together this gives:

$$A = LDL^T + R,$$

where

$$L = L_1 L_2 = \begin{pmatrix} I_\diamondsuit & 0 & 0 \\ A_{\heartsuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1} & I_\heartsuit & 0 \\ A_{\spadesuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1} & (A_{\spadesuit\heartsuit} - A_{\spadesuit\diamondsuit}\widetilde{D}_\diamondsuit^{-1}A_{\diamondsuit\heartsuit})\widetilde{D}_\heartsuit^{-1} & I_\spadesuit \end{pmatrix},\ D = A_2 = \begin{pmatrix} D_\diamondsuit & 0 & 0 \\ 0 & \widetilde{D}_\heartsuit & 0 \\ 0 & 0 & \hat{A}_\spadesuit \end{pmatrix}$$

and

$$R = R_0 + L_1 R_1 L_1^T = R_0 + R_1 = \begin{pmatrix} R_\diamondsuit & 0 & 0 \\ 0 & R_\heartsuit & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

It can be shown (see e.g., [5]) that the properties of the original matrix A are conserved, i.e., the matrices given by the 9-point stencils are also symmetric positive definite M-matrices. Therefore, the RRB-factorization algorithm can be reapplied, i.e., another RRB-iteration can be performed on the remainder $\hat{A}_\spadesuit$.

Full RRB versus RRB-ℓ

After a certain number of RRB-iterations only a single node remains. This is called full RRB. However, after each level of red-black numbering, lumping and elimination, one can already stop, say at level $\ell \leq \ell_{\max}$ (see Equation (2.1)). The remaining black nodes $B^{[\ell]}$ have a 9-point dependency structure and for this level a full Cholesky decomposition

$$M_\ell = L_\ell D_\ell L_\ell^T \tag{2.7}$$

is computed. This is called ℓ-step RRB, denoted by RRB-ℓ. By stopping earlier the factorization $A = LDL^T + R$ becomes more exact; however, in that case the construction of the Cholesky factor L requires more effort and L has more fill-in. One should strive for a good balance; we go on with the RRB-iterations until the size of the remaining system of equations becomes that small that the costs of solving (2.7) are about equal to, or at least not much larger, than the costs of the rest of the factorization.

Starting with a 5-point stencil

In case the RRB-factorization algorithm is applied to the matrix A as in system (1.1), which is given by a 5-point stencil, $A_\diamondsuit$ becomes a diagonal matrix and $A_{\heartsuit\spadesuit} = A_{\spadesuit\heartsuit^T} = 0$. The first elimination becomes exact and accordingly $R_\diamondsuit = 0$.

Algorithm

Pseudo code for the RRB-factorization algorithm is provided in Algorithm 2.

Example

In this example the RRB-factorization algorithm is applied to a matrix $A \in \mathbb{R}^{64\times 64}$ resulting from an 8×8 grid of unknowns, see Figure 5. The figure shows the effects of consecutive red-black orderings, lumping, and elimination of red nodes on the dependency structure and sparsity pattern of $L + D + L^T$.

Algorithm 2 The RRB-factorization algorithm starting with a 9-point stencil.

```
1. Choose the number of levels $\ell \leq \ell_{\max}$
2. Set $k = 0$
3. **While** $(k \leq \ell)$ **do**
4.    Apply red-black ordering: $R^{[k]}/B^{[k]}$
5.    Apply lumping procedure to $R^{[k]}$-nodes
6.    Eliminate $R^{[k]}$-nodes using 5-point stencils for $R^{[k]}$-nodes
7.    $k = k + 1$
8. **End while**
9. Make complete Cholesky decomposition for remaining 9-point stencil: $M_\ell = L_\ell D_\ell L_\ell^T$
```

2.3 The RRB-factorization algorithm used as a preconditioner in PCG

The matrix A is factorized as $A = LDL^T + R$, see Equation (2.2). As a preconditioner for the PCG-algorithm the matrix

$$M = LDL^T \approx A \tag{2.8}$$

is taken. The smaller the numbers in R in absolute value are, the better M resembles A. The combination of the PCG-algorithm and the RRB-factorization algorithm for construction of the precondioner M is called the RRB-solver.

By starting from a 5-point stencil, the elimination of $R^{[1]}$-nodes becomes exact. Hence, in this case PCG can be applied to the resulting 1st Schur complement S_1 instead of the entire matrix A. This can be seen as follows. By applying a red-black numbering we can write $A\mathbf{x} = \mathbf{b}$ as

$$\begin{bmatrix} D_r & A_{rb} \\ A_{br} & D_b \end{bmatrix} \begin{bmatrix} \mathbf{x}_r \\ \mathbf{x}_b \end{bmatrix} = \begin{bmatrix} \mathbf{b}_r \\ \mathbf{b}_b \end{bmatrix},$$

where the red nodes are indicated by the subscript 'r' and the black nodes by the subscript 'b'. Furthermore, D_b and D_r are diagonal matrices and $A_{rb} = A_{br}^T$ are matrices with four diagonals, see Figure 3a (or 5b). Applying Gaussian elimination, i.e., elimination of the red nodes, yields:

$$\begin{bmatrix} D_r & A_{rb} \\ 0 & S_1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_r \\ \mathbf{x}_b \end{bmatrix} = \begin{bmatrix} \mathbf{b}_r \\ \mathbf{b}_1 \end{bmatrix},$$

where

$$S_1 := D_b - A_{br} D_r^{-1} A_{rb} \tag{2.9}$$

is called the 1st Schur complement of A (given by a 9-point stencil) and

$$\mathbf{b}_1 := \mathbf{b}_b - A_{br} D_r^{-1} \mathbf{b}_r \tag{2.10}$$

is the corresponding RHS. Hence the original system (1.1) can alternatively be solved as follows:

1. Compute $\mathbf{b}_1$ using (2.10);

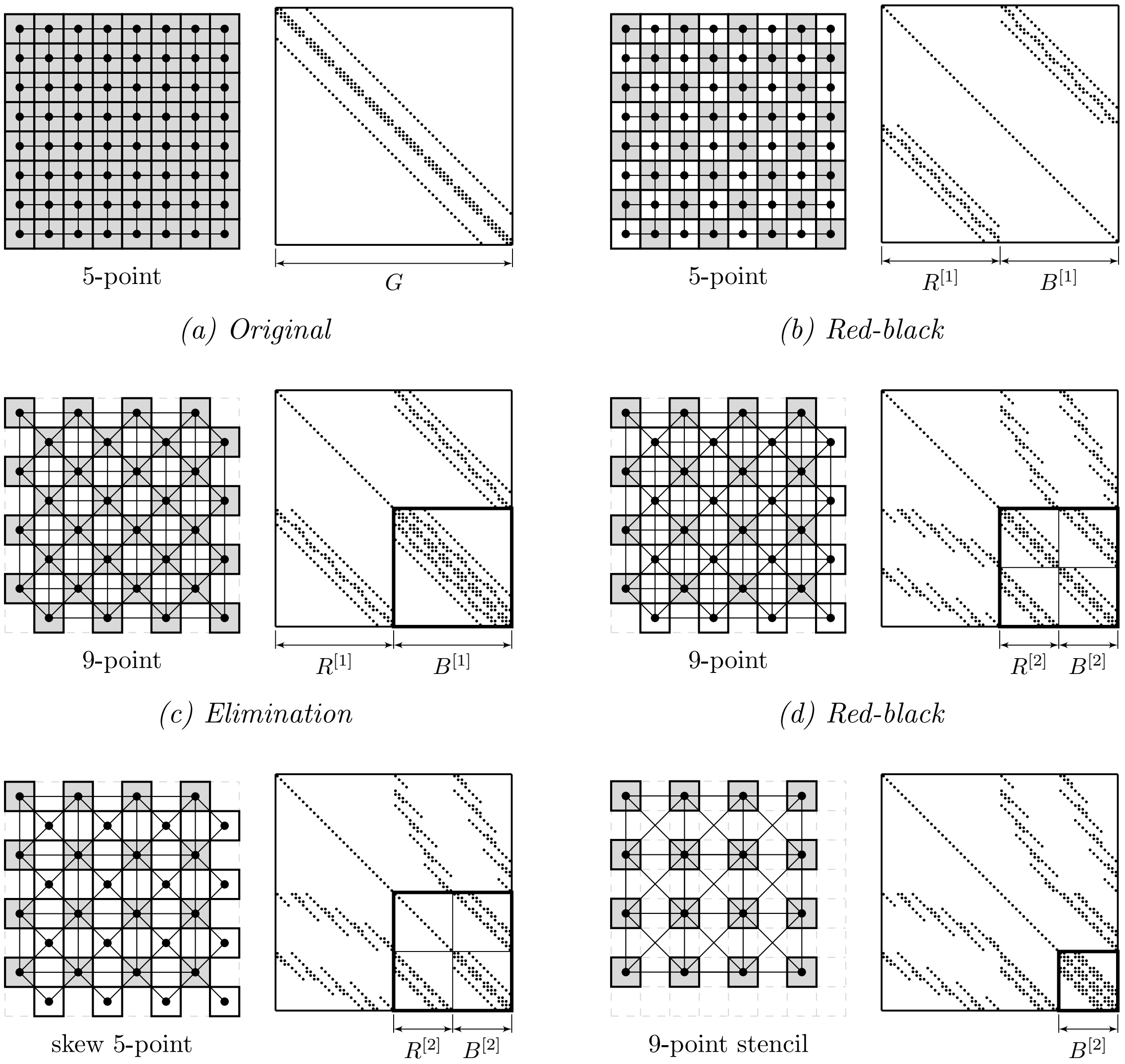


Figure 5: Effects of the RRB-factorization algorithm for $A \in \mathbb{A}^{64\times 64}$ with $\ell = 2$ on the dependency structure and the sparsity pattern of $L + D + L^T$. For the remaining $B^{[2]}$-nodes a full Cholesky factorization is computed.

2. Apply CG to the system
$$S_1 \mathbf{x}_b = \mathbf{b}_1, \tag{2.11}$$
where S_1 is the 1st Schur complement given by (2.9);

3. Compute $\mathbf{x}_r$ via $\mathbf{x}_r = D_r^{-1}(\mathbf{b}_r - A_{rb}\mathbf{x}_b)$.

Solving system (2.11) instead of the original system (1.1) is beneficial to the total amount of computational work. Since S_1 consists of only the $B^{[1]}$-nodes, the number of flops for computing the vector updates and inner products in the PCG-algorithm is reduced by a factor two. The number of flops in the matrix-vector product $\mathbf{q} = S_1\mathbf{p}$ remains the same as the matrix-vector product with A, because the matrix S_1 is now given by a 9-point stencil instead of by a 5-point stencil.

2.4 Other lumping procedures

In Section 2.2 it was explained how lumping procedures are used to reduce 9-point stencils to 5-point stencils. Afterwards, the obtained 5-point stencils were used to eliminate the red nodes at the current level. In this section other lumping strategies are discussed.

For $k = 1, 3, 5, \ldots$ consider:

$$\begin{bmatrix} \cdot & \cdot & N & \cdot & \cdot \\ \cdot & NW & \cdot & NE & \cdot \\ W & \cdot & C & \cdot & E \\ \cdot & SW & \cdot & SE & \cdot \\ \cdot & \cdot & S & \cdot & \cdot \end{bmatrix}^{[k]} \Longrightarrow \begin{bmatrix} NW & \cdot & NE \\ \cdot & \widetilde{C} & \cdot \\ SW & \cdot & SE \end{bmatrix}^{[k]},$$

where

$$\widetilde{C}^{[k]} = C^{[k]} + \omega\big(N^{[k]} + E^{[k]} + S^{[k]} + W^{[k]}\big) \tag{2.12}$$

and for $k = 0, 2, 4, \ldots$:

$$\begin{bmatrix} NW & \cdot & N & \cdot & NE \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ W & \cdot & C & \cdot & E \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ SW & \cdot & S & \cdot & SE \end{bmatrix}^{[k]} \Longrightarrow \begin{bmatrix} \cdot & N & \cdot \\ W & \widetilde{C} & E \\ \cdot & S & \cdot \end{bmatrix}^{[k]},$$

where

$$\widetilde{C}^{[k]} = C^{[k]} + \omega\big(NE^{[k]} + SE^{[k]} + SW^{[k]} + NW^{[k]}\big). \tag{2.13}$$

In both (2.12) and (2.13) $0 \leq \omega \leq 1$ is called the relaxation parameter. The easiest way is to just drop the unwanted fill-in and take $\omega = 0$. This leads to an incomplete Cholesky factorization. By choosing $\omega = 1$ the four most outer elements are lumped towards the main diagonal like we did in Section 2.2. In that case the row sum is maintained and modified incomplete Cholesky is obtained. A modified incomplete Cholesky decomposition has the property that the row sums of the residual matrix are zero. Hence the decomposition is exact for a constant vector. This feature improves the quality of the preconditioner.

Next one can choose to which nodes the lumping procedure is applied in a certain level: to the red nodes only or to both the red and the black nodes in the same level. In [5] Ciarlet investigated four of the indicated eight options, see Table 1. A final important observation is that the structure, i.e., the sparsity pattern, of the preconditioner does not change; for each of the different options the remainder on the black nodes is always given by a 9-point stencil.

Name preconditioner	ω	Nodes used
$M1u$	0	red
$M1m$	1	red
$M2u$	0	red and black
$M2m$	1	red and black

Table 1: Different lumping strategies (cf. [5]).

2.5 Spectral condition number of RRB-ℓ

As we have seen in Section 1, is the convergence rate of PCG dependent on the eigenvalues of $M^{-1}A$. Since the preconditioner M is an approximation of the system matrix A, the spectral condition number $\kappa(M^{-1}A)$ is expected to be smaller than $\kappa(A)$. This gives a sharper upperbound of the error (1.2) and therefore most likely a faster convergence.

In [2], [7] and [5] the RRB-ℓ preconditioner is investigated in detail for the Poisson problem with Dirichlet boundary conditions on a two-dimensional uniform grid with $n \times n$ unknowns and where n is of the form $n = 2^\ell - 1$. Different upper bounds can be found in the aforementioned literature. Notay [7] gives as an upper bound :

$$\kappa(M^{-1}A) \leq \frac{\sqrt{5}(\sqrt{5}-1)^{\ell-1}}{1+(-1)^\ell\left(\frac{3-\sqrt{5}}{2}\right)^{\ell-1}} \approx 1.8 \cdot 1.23^\ell.$$

Since the mesh spacing $h = 1/(n+1)$ and $\ell = \log_2(1/n) \approx \log_2 h^{-1}$, alternatively,

$$\kappa(M^{-1}A) \leq 1.8\, h^{-0.306}.$$

2.6 Application of the preconditioner

At each iteration of the PCG-algorithm, we have the preconditioner step $M\mathbf{z} = \mathbf{r}$ which needs to be solved for $\mathbf{z}$. Since the preconditioner M is factorized as $M = LDL^T$, see Equation (2.8), the system $M\mathbf{z} = \mathbf{r}$ can be solved efficiently in three steps: set $\mathbf{y} := L^T\mathbf{z}$ and $\mathbf{x} := DL^T\mathbf{z} = D\mathbf{y}$, then:

1. solve $\mathbf{x}$ from $L\mathbf{x} = \mathbf{r}$ using forward substitution;
2. compute $\mathbf{y} = D^{-1}\mathbf{x}$;
3. solve $\mathbf{z}$ from $L^T\mathbf{z} = \mathbf{y}$ using backward substitution.

Each of the three steps is computationally cheap.

Algorithm

For memory efficiency a single vector $\mathbf{z}$ is used instead of the three vectors $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$. Moreover, if $\ell \leq \ell_{\max}$ then at the final level ℓ a full Cholesky factorisation (2.7) is constructed for the remaining level of black nodes $B^{[\ell]}$. Pseudo code is provided below, see Algorithm 3.

Algorithm 3 Application of the RRB preconditioner.

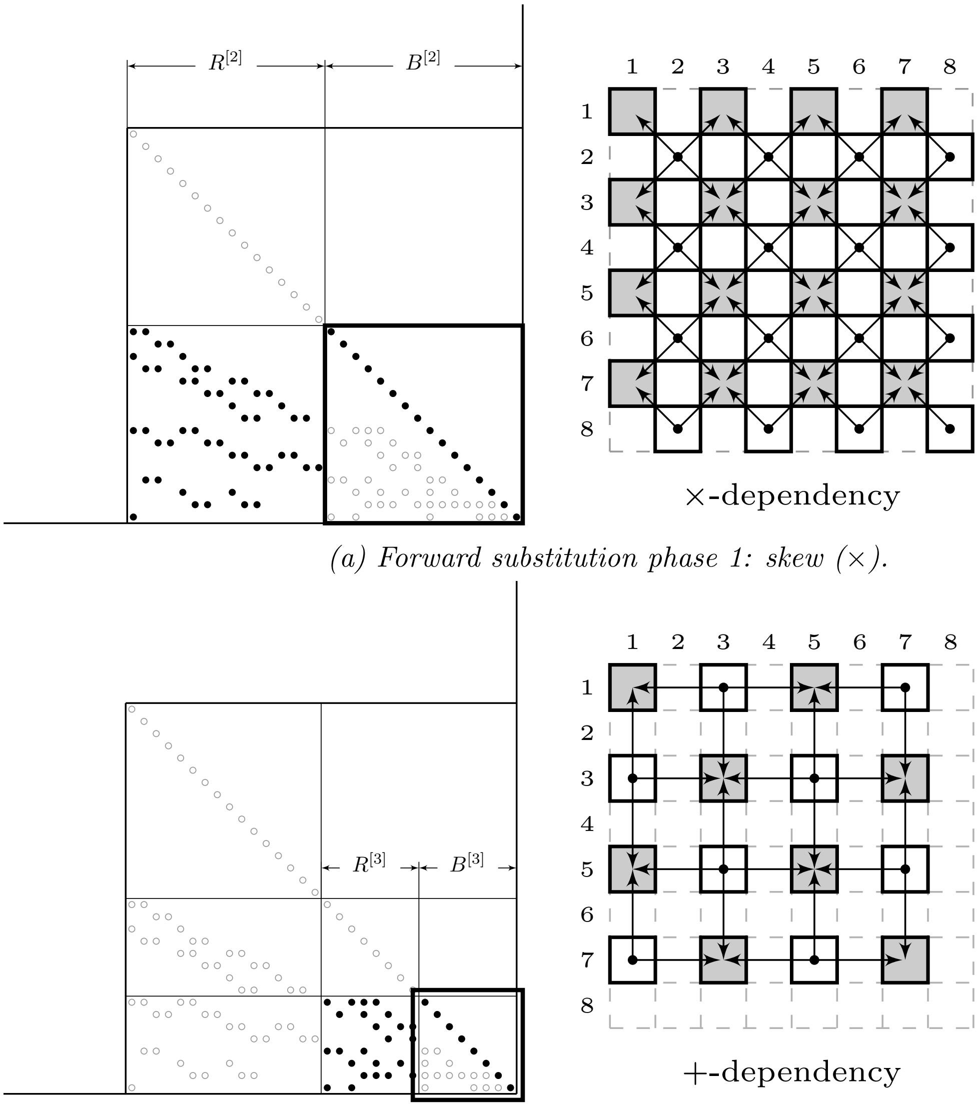
```
Given number of levels $\ell \leq \ell_{\max}$
Set $k = 2$
While ($k \leq \ell$) do                                  % forward substitution
    Update $\mathbf{z}$-values at $B^{[k]}$-nodes using $R^{[k]}$-nodes          % using skew 5-point ($\times$)
    If ($k + 1 == \ell$) then
        break
    End if
    Update $\mathbf{z}$-values at $B^{[k+1]}$-nodes using $R^{[k+1]}$-nodes      % using straight 5-point ($+$)
    $k = k + 2$
End while
Update $\mathbf{x}$-values at $B^{[1]}$-nodes                                    % diagonal scaling
Solve $L_\ell D_\ell L_\ell^T \mathbf{x}_\ell = \mathbf{z}_\ell$ and update $\mathbf{x}$-values in level $\ell$   % final level exact
Set $k = \ell$
While ($k \geq 2$) do                                     % backward substitution
    Update $\mathbf{z}$-values at $R^{[k]}$-nodes using $B^{k]}$-nodes           % using straight 5-point ($+$)
    If ($k - 1 == 2$) then
        break
    End if
    Update $\mathbf{z}$-values at $R^{[k-1]}$-nodes using $B^{[k-1]}$-nodes      % using skew 5-point ($\times$)
    $k = k - 2$
End while
```

Example

For a matrix $A \in \mathbb{R}^{64\times 64}$ resulting from an 8×8 grid of unknowns the forward substitution steps are visualized in Figure 6a and Figure 6b. As indicated by Algorithm 3 both the forward and backward substitution are performed level-wise in two phases. First, the $\mathbf{z}$-values at $B^{[k]}$-nodes are updated using the skew 5-point stencils ($\times$) at the $R^{[k]}$-nodes, see Figure 6a. Second, the $\mathbf{z}$-values at $B^{[k+1]}$-nodes are updated using the straight 5-point stencils ($+$) at the $R^{[k+1]}$-nodes, see Figure 6b. The backward substitution is done level-wise in the same manner, yet in the reverse order.

3 Parallel implementation of the RRB-solver

All the basic operations in the CG-algorithm, see Algorithm 1 (the vector updates, the inner products, and the matrix-vector product, in our case $\mathbf{q} = S_1\mathbf{p}$, where S_1 is the first Schur complement), can easily be parallelized on shared memory machines [6]. Secondly, the construction of the preconditioner, i.e., $M = LDL^T$, can be performed level-wise in parallel on shared memory machines. This can be seen from Algorithm 2: at each level each of the operations lumping, elimination, and substitution can be performed fully in parallel. Finally, the preconditioner step, i.e., solving $M\mathbf{z} = \mathbf{r}$ for $\mathbf{z}$, can be performed *level-wise*



(a) Forward substitution phase 1: skew (×).

(b) Forward substitution phase 2: straight (+)

Figure 6: Application of the RRB preconditioner.

in parallel on shared memory machines as well. This can be seen from Algorithm 3 as well as from Figure 6: at each level the gray nodes can be updated fully in parallel. In Figure 7 the gray areas indicate the blocks where fill-in has been lumped, and where the computations can be done fully in parallel.

It is however not possible to compute the different levels (the gray blocks) in parallel since forward and backward substitution are, at least to some extent, inherently sequential. Since the number of unknowns decreases fast, namely by a factor 2 per level, so does the amount of work, and therefore this is not a big issue. All together, we can conclude that the RRB-solver can be parallelized well on shared memory machines.

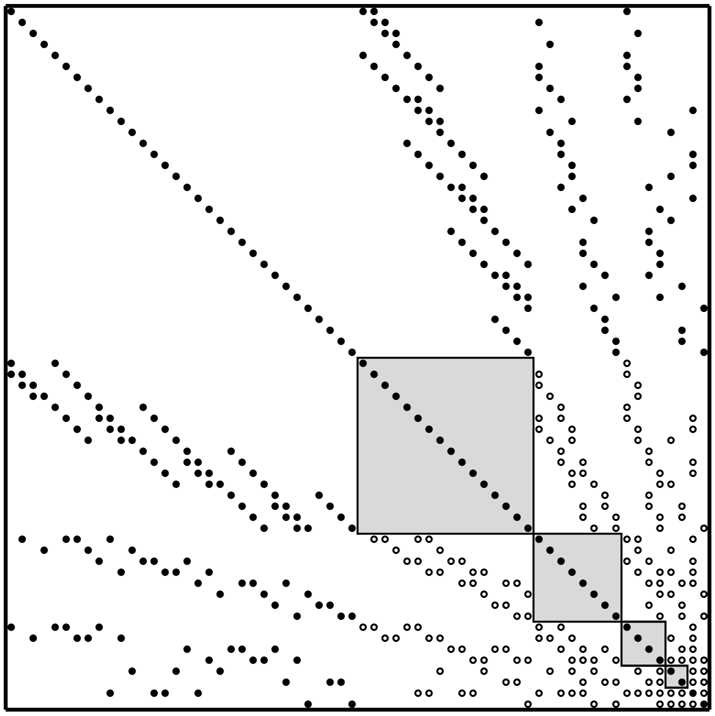


Figure 7: Sparsity pattern of $L + D + L^T$. The gray areas indicate parallel blocks.

3.1 A key idea to maximise bandwidth

The RRB-solver consists of level 1 and level 2 BLAS routines and is therefore a memory-intensive algorithm, i.e., the ratio

$$r = \frac{\#\text{memory operations}}{\#\text{floating point operations}}$$

is large. For small problem sizes already, the problem is already too big to fit in the (fast) cache of most modern computers[1]. Hence throughout the RRB-algorithm most data must be read from and written to the global memory over and over again. In order to obtain an efficient and fast solver we must deal with the issue of achieving good global memory throughput throughout the algorithm.

Achieving high global memory throughput is certainly not trivial for the RRB-solver because of the repeated red-black orderings. This can be seen from Figure 2. A naive (basic, rectangular, natural, lexicographic) storage of the data would result in a solver in which all the data required for the vector-updates, inner products and the matrix-vector product would be read from and written to the global memory with a stride of two. This follows from the fact that the basic routines of the RRB-solver operate on the $B^{[1]}$-nodes

[1] Example: Solving a 2D poisson problem consisting of 1000 × 1000 unknowns with the RRB-solver requires roughly 19 vectors of length 1 million hence 159 MB of data in case of double-precision. This is already much larger than the last level cache size of any modern architecture.

only. Even worse, during application of the preconditioner step, i.e., solving $M\mathbf{z} = \mathbf{r}$ for $\mathbf{z}$, the data would be accessed with increasing stride: $2, 4, 8, 16, \ldots$. It is a well-known fact that reading data from and writing data to the global memory with a stride leads to a waste of performance.

In order to obtain maximal throughput throughout the algorithm, for the first few (the finest) levels a different storage scheme is proposed, see Figure 8. This storage scheme is called the $r_1/r_2/b_1/b_2$-storage scheme [4].

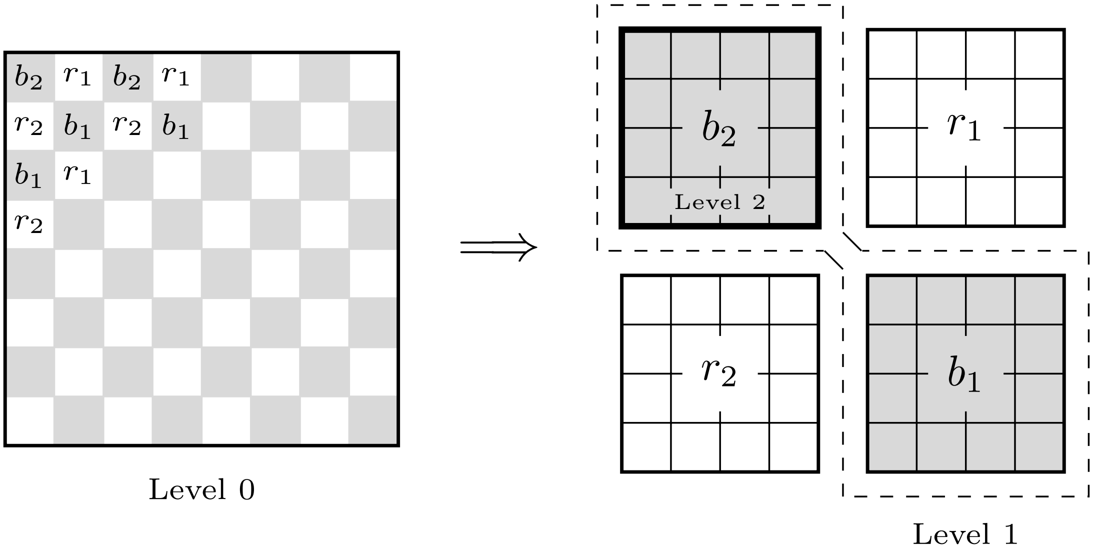


Figure 8: The $r_1/r_2/b_1/b_2$ storage scheme. The b_1- and b_2-nodes together form the intermediate levels with skew meshes ($\times$). The b_2-nodes form the next coarser level ($+$).

We see that the first level red nodes are divided into r_1- and r_2-nodes and the first level black nodes are divided into b_1- and b_2-nodes. The first important observation is now that, by storing the $B^{[1]}$ nodes into these two new arrays of b_1- and b_2-nodes, the data can always be accessed in a coalesced manner, without a stride.

The second important observation is that the b_2-nodes form the next coarser grid, and that, on this coarser grid, the $r_1/r_2/b_1/b_2$-storage scheme can be reapplied. In general, the odd levels k are stored into $b_1^{[k]}$- and $b_2^{[k]}$-nodes, and the even levels k are stored into the $b_2^{[k]}$-nodes.

The data used by the RRB-solver is stored using the $r_1/r_2/b_1/b_2$ storage scheme as much as possible: the 5 vectors in the PCG-algorithm $(\mathbf{r}, \mathbf{x}, \mathbf{z}, \mathbf{p}, \mathbf{q})$, see Algorithm 1, the vectors that describe the first Schur complement S_1 and the vectors that describe the preconditioner matrix M.

The matrix M and the vector $\mathbf{z}$ occuring in the preconditioner step $M\mathbf{z} = \mathbf{r}$ are stored in a recursive $r_1/r_2/b_1/b_2$-storage scheme which is at most only $1 + 1/4 + 1/16 + \ldots = 4/3$ times as expensive as the naive storage. From Figure 6 it can be seen that the $r_1/r_2/b_1/b_2$-storage scheme can be used virtually for free during the preconditioner step $M\mathbf{z} = \mathbf{r}$. This can be seen as follows. In case of application of the preconditioner, the updated $\mathbf{z}$-values at level k are directly written into the $r_1/r_2/b_1/b_2$-arrays of level $k+1$ in case of forward substitution and vice versa in case of backward subsitution.

3.2 Implementation details

In this section some details are given to implement the $r_1/r_2/b_1/b_2$ storage format in a good manner. For optimal performance we introduce a format that allows for aligned memory transfers, vectorization and which makes if-statements[2] unnecessary. The format is shown in Figure 9 in which a problem of $N_x \times N_y = 40 \times 75$ unknowns is taken as an example. The following can be seen:

1. The 40×75 nodes are divided into r_1-, r_2-, b_1- and b_2-nodes. Notice the slight difference in number of r_1- and r_2-nodes, namely 20×33 versus 20×32. This is due to the odd number of nodes in y-direction, namely 75. In x-direction there is no difference in size. Similarly, there is a difference in number of b_1- and b_2-nodes.

2. Four *compute areas* indicated by the four bold rectangles, each of size $cx[0] \times cy[0]$. Computations are only done directly for the elements that lie within one of the compute areas. The elements that lie within the borders are not used or only used indirectly because of the 5-point stencils. For example, consider the forward substitution Phase 1 in the application of the preconditioner, see Figure 6a. Clearly, during this phase, the b_1- and b_2-nodes are updated using the r_1- and r_2-nodes according to a skew ($\times$) 5-point stencil. A double for-loop runs over the entire compute areas b_1 and b_2 and data is read from areas r_1 and r_2.

3. The entire grid consists of a collection of uniformly sized *compute blocks*. Here the compute blocks are shown as squares of size $dcb \times dcb = 16 \times 16$. The compute blocks may also be rectangular and they may also have a different size than the width borders. In C, C++ or CUDA C, which are examples of row-major oriented languages, the size in x-direction is important. dcb should preferably be a multiple of 8 (double-precision) or 16 (single-precision) to meet the typical 64-byte cache lines of modern computing devices and allowing for aligned storage and vectorization. For row-major oriented programming languages, the size of the compute block in y-direction is less important. In Fortran or Fortran CUDA, which are examples of column-major oriented languages, it is the other way around: the size in y-direction is important and the size in x-direction is less important.

4. The data is padded. There is a border of width dcb around each of the four blocks of nodes. In this figure: $dcb = 16$. The entire $r_1/r_2/b_1/b_2$ grid has size $nx[0] \times ny[0]$. The border ensures that all data is aligned and that no if-statements are needed: the edge elements of the original problem (light gray) are no longer edge elements but belong to the inside of the larger array of size $nx[0] \times ny[0]$.

[2] With a naive storage format if-statements are necessary for the computations along the edges of the domain to prevent for 'out-of-index' runtime errors.

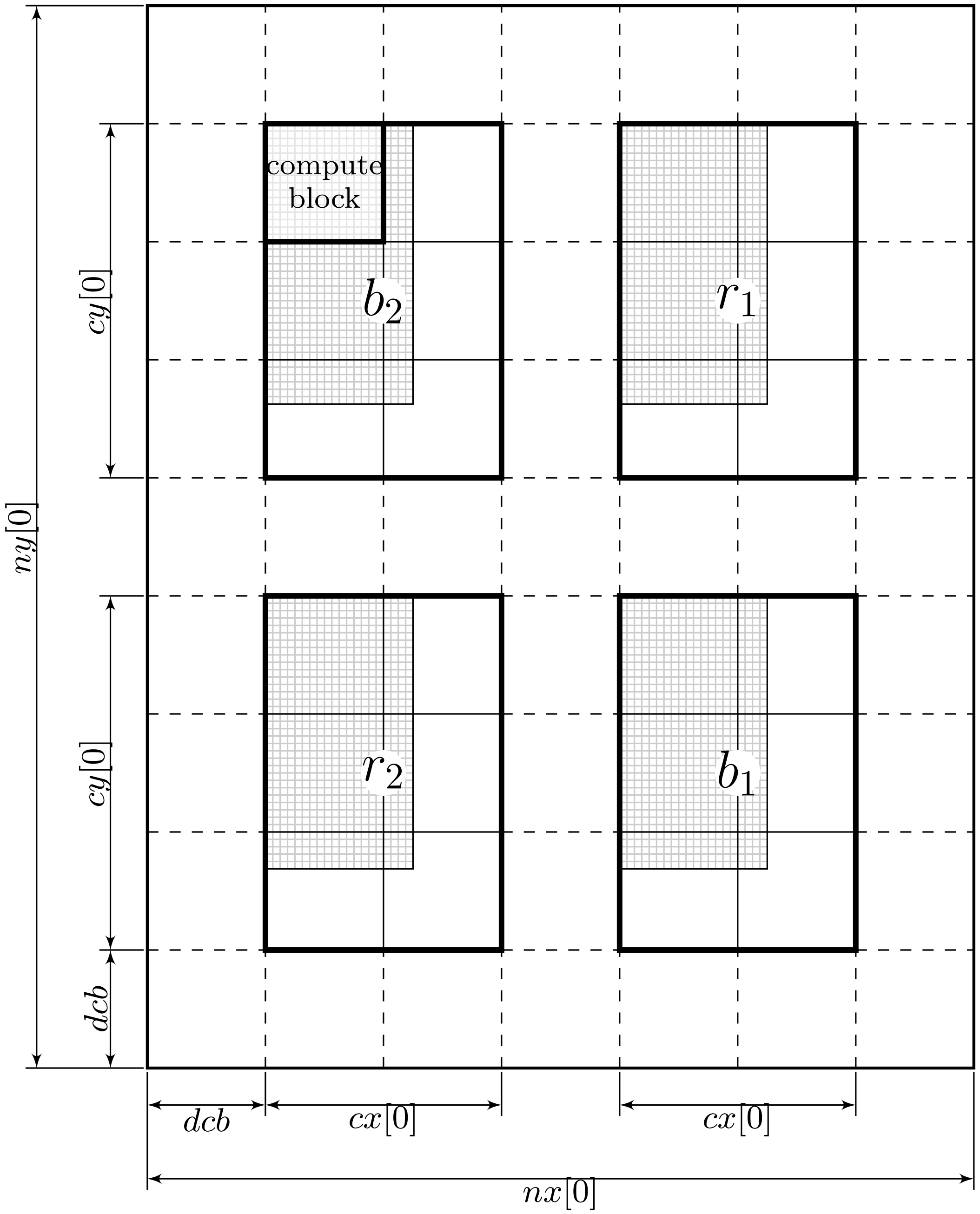


Figure 9: The $r_1/r_2/b_1/b_2$ storage scheme for a problem of 40×75 unknowns.

Next we introduce, similar to the desired number of RRB-levels ℓ, another adjustable parameter, namely g: the desired number of $r_1/r_2/b_1/b_2$ grids. In this way one can balance the algorithm between: (i) optimal memory transfers on the $r_1/r_2/b_1/b_2$ storage format on the first few (the finest) levels with many unknowns and (ii) the advantage of the (fast) cache on the last (the coarsest) levels with few unknowns. The motivation for this is that from a certain point on the remaining unknowns may fit entirely in the cache of the computing hardware.

The maximum number of $r_1/r_2/b_1/b_2$ grids is denoted by $g_{\max}$ and is dependent on: (i) the maximal number of levels $\ell_{\max}$, and (ii) the size of the compute blocks. Similar to Equation (2.1) one can derive:

$$g_{\max} = \lceil \log_2(\max\{N_x, N_y\}/dcb) \rceil, \tag{3.1}$$

where *dcb* is the size of the square compute block as mentioned above.

Example

For a problem with $N_x \times N_y = 411 \times 277$ unknowns and compute block with size $dcb \times dcb = 32 \times 32$, we have $\ell_{\max} = 2\lceil \log_2(\max\{411, 277\}) \rceil + 1 = 19$, see Equation (2.1), and $g_{\max} = \lceil \log_2(\max\{411, 277\}/32) \rceil = 4$, see Equation (3.1). Hence the computations for levels $1, 2, \ldots, 8$ are performed with the $r_1/r_2/b_1/b_2$ storage format, and the computations for the remaining 11 levels $9, 10, \ldots, 19$ are performed with the naive storage format.

References

[1] O. Axelsson and V. Eijkhout. The nested recursive two-level factorization method for nine-point difference matrices. *Journal on Scientific and Statistical Computing*, 12:1373–1400, 1991.

[2] C. Brand. An incomplete-factorization preconditioning using repeated red-black ordering. *Numerische Mathematik*, 61:433–454, 1992.

[3] C. Brand and Z. Heinemann. A new iterative solution technique for reservoir simulation equations on locally refined grids. *SPE Reservoir Engineering*, 5:555–560, 1990.

[4] M. de Jong, A. van der Ploeg, A. Ditzel, and C. Vuik. Real-time computation of interactive waves on the GPU. In *15th Numerical Towing Tank Symposium*, pages 111–116, Cortona, Italy, 2012.

[5] P. Ciarlet Jr. Repeated red-black ordering: a new approach. *Numerical Algorithms*, 7:295–324, 1994.

[6] R. Li and Y. Saad. GPU-accelerated preconditioned iterative linear solvers. *The Journal of Supercomputing*, 63(2):443–466, 2013.

[7] Y. Notay and Z. Ould Amar. A nearly optimal preconditioning based on recursive red-black orderings. *Numerical Linear Algebra with Applications*, 4:369–391, 1997.

[8] Y. Saad. *Iterative methods for sparse linear systems.* pub-SIAM, second edition, 2003.

[9] H. van der Vorst. *Iterative Krylov Methods for Large Linear Systems.* University Press, 2003.