$$A = \begin{pmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{pmatrix} \tag{7.20}$$

for the matrix elements and starting with $u_1 = b_1$, these computations are

$$\begin{aligned} l_{k+1} &= a_{k+1}/u_k, \\ u_{k+1} &= b_{k+1} - l_{k+1} \cdot c_k \ . \end{aligned} \tag{7.21}$$

After $n - 1$ steps an LU decomposition $A = LU$ of matrix (7.20) with

$$L = \begin{pmatrix} 1 & & 0 \\ l_2 & 1 & \\ & \ddots & \ddots & \\ 0 & & l_n & 1 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} u_1 & c_1 & & 0 \\ & \ddots & \ddots & \\ & & u_{n-1} & c_{n-1} \\ 0 & & & u_n \end{pmatrix}$$

results. The right-hand side y is transformed correspondingly according to

$$\tilde{y}_{k+1} = y_{k+1} - l_{k+1} \cdot \tilde{y}_k \ .$$

The solution x is computed from the upper triangular matrix U by a backward substitution, starting with $x_n = \tilde{y}_n/u_n$ and solving the equations $u_i x_i + c_i x_{i+1} = \tilde{y}_i$ one after another resulting in

$$x_i = \frac{\tilde{y}_i}{u_i} - \frac{c_i}{u_i} x_{i+1} \quad \text{for } i = n-1, \ldots, 1 \ .$$

The computational complexity of the Gaussian elimination is reduced to $O(n)$ for tridiagonal systems. However, the elimination phase computing l_k and u_k according to Eq. (7.21) is inherently sequential, since the computation of l_{k+1} depends on u_k and the computation of u_{k+1} depends on l_{k+1}. Thus, in this form the Gaussian elimination or LU decomposition has to be computed sequentially and is not suitable for a parallel implementation.

7.2.2.2 Recursive Doubling for Tridiagonal Systems

An alternative approach for solving a linear equation system with tridiagonal matrix is the method of **recursive doubling** or **cyclic reduction**. The methods of recursive doubling and cyclic reduction also use elimination steps but contain potential parallelism [72, 71]. Both techniques can be applied if the coefficient matrix is either symmetric and positive definite or diagonal dominant [115]. The elimination steps in both methods are applied to linear equation systems $Ax = y$ with the matrix structure shown in (7.20), i.e.,

$$\begin{aligned} b_1\, x_1 + c_1\, x_2 &= y_1, \\ a_i\, x_{i-1} + b_i\, x_i + c_i\, x_{i+1} &= y_i \quad \text{for } i = 2, \ldots, n-1, \\ a_n\, x_{n-1} + b_n\, x_n &= y_n \quad . \end{aligned}$$

The method, which was first introduced by Hockney and Golub in [91], uses two equations $i - 1$ and $i + 1$ to eliminate the variables x_{i-1} and x_{i+1} from equation i. This results in a new equivalent equation system with a coefficient matrix with three non-zero diagonals where the diagonals are moved to the outside. Recursive doubling and cyclic reduction can be considered as two implementation variants for the same numerical idea of the method of Hockney and Golub. The implementation of recursive doubling repeats the elimination step, which finally results in a matrix structure in which only the elements in the principal diagonal are non-zero and the solution vector x can be computed easily. Cyclic reduction is a variant of recursive doubling which also eliminates variables using neighboring rows. But in each step the elimination is only applied to half of the equations and, thus, less computations are performed. On the other hand, the computation of the solution vector x requires a substitution phase.

We would like to mention that the terms recursive doubling and cyclic reduction are used in different ways in the literature. Cyclic reduction is sometimes used for the numerical method of Hockney and Golub in both implementation variants, see [60, 115]. On the other hand the term recursive doubling (or full recursive doubling) is sometimes used for a different method, the method of Stone [168]. This method applies the implementation variants sketched above in Eq. (7.21) resulting from the Gaussian elimination, see [61, 173]. In the following, we start the description of recursive doubling for the method of Hockney and Golub according to [61] and [13].

Recursive doubling considers three neighboring equations $i - 1, i, i + 1$ of the equation system $Ax = y$ with coefficient matrix A in the form (7.20) for $i = 3, 4, \ldots, n - 2$. These equations are

$$\begin{aligned} a_{i-1}x_{i-2} + b_{i-1}x_{i-1} + c_{i-1}x_i \qquad\qquad\qquad\qquad &= y_{i-1}, \\ a_i x_{i-1} \quad + b_i x_i \quad + c_i x_{i+1} \qquad\qquad &= y_i, \\ a_{i+1}x_i + b_{i+1}x_{i+1} + c_{i+1}x_{i+2} &= y_{i+1}. \end{aligned}$$

Equation $i - 1$ is used to eliminate x_{i-1} from the ith equation and equation $i + 1$ is used to eliminate x_{i+1} from the ith equation. This is done by reformulating equations $i - 1$ and $i + 1$ to

$$\begin{aligned} x_{i-1} &= \frac{y_{i-1}}{b_{i-1}} - \frac{a_{i-1}}{b_{i-1}} x_{i-2} - \frac{c_{i-1}}{b_{i-1}} x_i \ , \\ x_{i+1} &= \frac{y_{i+1}}{b_{i+1}} - \frac{a_{i+1}}{b_{i+1}} x_i - \frac{c_{i+1}}{b_{i+1}} x_{i+2} \end{aligned}$$

and inserting those descriptions of x_{i-1} and x_{i+1} into equation i. The resulting new equation i is

$$a_i^{(1)} x_{i-2} + b_i^{(1)} x_i + c_i^{(1)} x_{i+2} = y_i^{(1)} \tag{7.22}$$

with coefficients

$$\begin{aligned} a_i^{(1)} &= \alpha_i^{(1)} \cdot a_{i-1}, \\ b_i^{(1)} &= b_i + \alpha_i^{(1)} \cdot c_{i-1} + \beta_i^{(1)} \cdot a_{i+1}, \\ c_i^{(1)} &= \beta_i^{(1)} \cdot c_{i+1}, \\ y_i^{(1)} &= y_i + \alpha_i^{(1)} \cdot y_{i-1} + \beta_i^{(1)} \cdot y_{i+1}, \end{aligned} \tag{7.23}$$

and

$$\begin{aligned} \alpha_i^{(1)} &:= -a_i / b_{i-1} \ , \\ \beta_i^{(1)} &:= -c_i / b_{i+1} \ . \end{aligned}$$

For the special cases $i = 1, 2, n-1, n$, the coefficients are given by

$$\begin{aligned} & b_1^{(1)} = b_1 + \beta_1^{(1)} \cdot a_2, \quad && y_1^{(1)} = y_1 + \beta_1^{(1)} \cdot y_2, \\ & b_n^{(1)} = b_n + \alpha_n^{(1)} \cdot c_{n-1}, && y_n^{(1)} = b_n + \alpha_n^{(1)} \cdot y_{n-1}, \\ & a_1^{(1)} = a_2^{(1)} = 0, \text{ and} && c_{n-1}^{(1)} = c_n^{(1)} = 0 \ . \end{aligned}$$

The values for $a_{n-1}^{(1)}$, $a_n^{(1)}$, $b_2^{(1)}$, $b_{n-1}^{(1)}$, $c_1^{(1)}$, $c_2^{(1)}$, $y_2^{(1)}$, and $y_{n-1}^{(1)}$ are defined as in Eq. (7.23). Equation (7.22) forms a linear equation system $A^{(1)}x = y^{(1)}$ with a coefficient matrix

$$A^{(1)} = \begin{pmatrix} b_1^{(1)} & 0 & c_1^{(1)} & & & 0 \\ 0 & b_2^{(1)} & 0 & c_2^{(1)} & & \\ a_3^{(1)} & 0 & b_3^{(1)} & \ddots & \ddots & \\ & a_4^{(1)} & \ddots & \ddots & \ddots & c_{n-2}^{(1)} \\ & & \ddots & \ddots & \ddots & 0 \\ 0 & & & a_n^{(1)} & 0 & b_n^{(1)} \end{pmatrix}.$$

Comparing the structure of $A^{(1)}$ with the structure of A, it can be seen that the diagonals are moved to the outside.

In the next step, this method is applied to the equations $i-2, i, i+2$ of the equation system $A^{(1)}x = y^{(1)}$ for $i = 5, 6, \ldots, n-4$. Equation $i-2$ is used to eliminate x_{i-2} from the ith equation and equation $i+2$ is used to eliminate x_{i+2} from the ith equation. This results in a new ith equation

$$a_i^{(2)} x_{i-4} + b_i^{(2)} x_i + c_i^{(2)} x_{i+4} = y_i^{(2)},$$

which contains the variables x_{i-4}, x_i, and x_{i+4}. The cases $i = 1, \ldots, 4, n-3, \ldots, n$ are treated separately as shown for the first elimination step. Altogether a next equation system $A^{(2)}x = y^{(2)}$ results in which the diagonals are further moved to the outside. The structure of $A^{(2)}$ is

$$A^{(2)} = \begin{pmatrix} b_1^{(2)} & 0 & 0 & 0 & c_1^{(2)} & & & 0 \\ 0 & b_2^{(2)} & & & & c_2^{(2)} & & \\ 0 & & \ddots & & & & \ddots & \\ 0 & & & \ddots & & & & c_{n-4}^{(2)} \\ a_5^{(2)} & & & & \ddots & & & 0 \\ & a_6^{(2)} & & & & \ddots & & 0 \\ & & \ddots & & & & \ddots & 0 \\ 0 & & & a_n^{(2)} & 0 & 0 & 0 & b_n^{(2)} \end{pmatrix} .$$

The following steps of the recursive doubling algorithm apply the same method to the modified equation system of the last step. Step k transfers the side diagonals $2^k - 1$ positions away from the main diagonal, compared to the original coefficient matrix. This is reached by considering equations $i - 2^{k-1}, i, i + 2^{k-1}$:

$$\begin{aligned} a_{i-2^{k-1}}^{(k-1)} x_{i-2^k} + b_{i-2^{k-1}}^{(k-1)} x_{i-2^{k-1}} + c_{i-2^{k-1}}^{(k-1)} x_i &= y_{i-2^{k-1}}^{(k-1)}, \\ a_i^{(k-1)} x_{i-2^{k-1}} + b_i^{(k-1)} x_i + c_i^{(k-1)} x_{i+2^{k-1}} &= y_i^{(k-1)}, \\ a_{i+2^{k-1}}^{(k-1)} x_i + b_{i+2^{k-1}}^{(k-1)} x_{i+2^{k-1}} + c_{i+2^{k-1}}^{(k-1)} x_{i+2^k} &= y_{i+2^{k-1}}^{(k-1)}. \end{aligned}$$

Equation $i - 2^{k-1}$ is used to eliminate $x_{i-2^{k-1}}$ from the ith equation and equation $i + 2^{k-1}$ is used to eliminate $x_{i+2^{k-1}}$ from the ith equation. Again, the elimination is performed by computing the coefficients for the next equation system. These coefficients are

$$\begin{aligned} a_i^{(k)} &= \alpha_i^{(k)} \cdot a_{i-2^{k-1}}^{(k-1)} \quad \text{for } i = 2^k + 1, \ldots, n, \text{ and } a_i^{(k)} = 0 \text{ otherwise}, \\ c_i^{(k)} &= \beta_i^{(k)} \cdot c_{i+2^{k-1}}^{(k-1)} \quad \text{for } i = 1, \ldots, n - 2^k, \text{ and } c_i^{(k)} = 0 \text{ otherwise}, \\ b_i^{(k)} &= \alpha_i^{(k)} \cdot c_{i-2^{k-1}}^{(k-1)} + b_i^{(k-1)} + \beta_i^{(k)} \cdot a_{i+2^{k-1}}^{(k-1)} \quad \text{for } i = 1, \ldots, n\ , \\ y_i^{(k)} &= \alpha_i^{(k)} \cdot y_{i-2^{k-1}}^{(k-1)} + y_i^{(k-1)} + \beta_i^{(k)} \cdot y_{i+2^{k-1}}^{(k-1)} \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{7.24}$$

with

$$\begin{aligned} \alpha_i^{(k)} &:= -a_i^{(k-1)} / b_{i-2^{k-1}}^{(k-1)} \quad \text{for } i = 2^{k-1} + 1, \ldots, n\ , \\ \beta_i^{(k)} &:= -c_i^{(k-1)} / b_{i+2^{k-1}}^{(k-1)} \quad \text{for } i = 1, \ldots, n - 2^{k-1}\ . \end{aligned} \tag{7.25}$$

The modified equation i results by multiplying equation $i - 2^{k-1}$ from step $k - 1$ with $\alpha_i^{(k)}$, multiplying equation $i + 2^{k-1}$ from step $k - 1$ with $\beta_i^{(k)}$, and adding both to equation i. The resulting ith equation is

$$a_i^{(k)} x_{i-2^k} + b_i^{(k)} x_i + c_i^{(k)} x_{i+2^k} = y_i^{(k)} \tag{7.26}$$

with the coefficients (7.24). The cases $k = 1, 2$ are special cases of this formula. The initialization for $k = 0$ is the following:

$$\begin{aligned} a_i^{(0)} &= a_i && \text{for } i = 2, \ldots, n \ , \\ b_i^{(0)} &= b_i && \text{for } i = 1, \ldots, n \ , \\ c_i^{(0)} &= c_i && \text{for } i = 1, \ldots, n-1 \ , \\ y_i^{(0)} &= y_i && \text{for } i = 1, \ldots, n \ . \end{aligned}$$

and $a_1^{(0)} = 0$, $c_n^{(0)} = 0$. Also, for the steps $k = 0, \ldots, \lceil \log n \rceil$ and $i \in \mathbb{Z} \setminus \{1, \ldots, n\}$ the values

$$\begin{aligned} a_i^{(k)} = c_i^{(k)} = y_i^{(k)} &= 0 \ , \\ b_i^{(k)} &= 1 \ , \\ x_i &= 0 \end{aligned}$$

are set. After $N = \lceil \log n \rceil$ steps, the original matrix A is transformed into a diagonal matrix $A^{(N)}$

$$A^{(N)} = \text{diag}(b_1^{(N)}, \ldots, b_n^{(N)})$$

in which only the main diagonal contains non-zero elements. The solution x of the linear equation system can be directly computed using this matrix and the correspondingly modified vector $y^{(N)}$:

$$x_i = y_i^{(N)} / b_i^{(N)} \text{ for } i = 1, 2, \ldots, n \ .$$

To summarize, the recursive doubling algorithm consists of two main phases:

1. Elimination phase: Compute the values $a_i^{(k)}$, $b_i^{(k)}$, $c_i^{(k)}$, and $y_i^{(k)}$ for $k = 1, \ldots, \lceil \log n \rceil$ and $i = 1, \ldots, n$ according to Eqs. (7.24) and (7.25).
2. Solution phase: Compute $x_i = y_i^{(N)} / b_i^{(N)}$ for $i = 1, \ldots, n$ with $N = \lceil \log n \rceil$.

The first phase consists of $\lceil \log n \rceil$ steps where in each step $O(n)$ values are computed. The sequential asymptotic runtime of the algorithm is therefore $O(n \cdot \log n)$ which is asymptotically slower than the $O(n)$ runtime for the Gaussian elimination approach described earlier. The advantage is that the computations in each step of the elimination and the substitution phase are independent and can be performed in parallel. Figure 7.10 illustrates the computations of the recursive doubling algorithm and the data dependencies between different steps.

7.2.2.3 Cyclic Reduction for Tridiagonal Systems

The recursive doubling algorithm offers a large degree of potential parallelism but has a larger computational complexity than the Gaussian elimination caused by

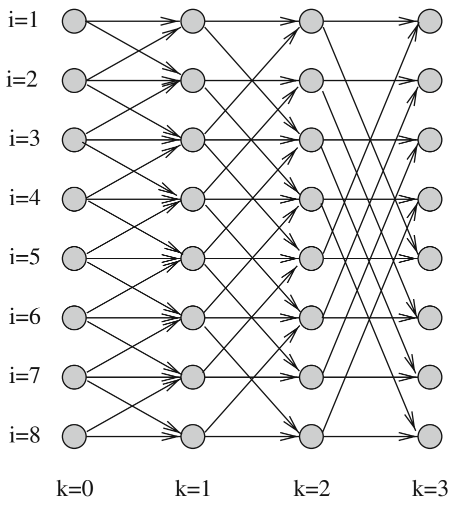

Fig. 7.10 Dependence graph for the computation steps of the recursive doubling algorithm in the case of three computation steps and eight equations. The computations of step k are shown in column k of the illustration. Column k contains one node for each equation i, thus representing the computation of all coefficients needed in step k. Column 0 represents the data of the coefficient matrix of the linear system. An edge from a node i in step k to a node j in step $k+1$ means that the computation at node j needs at least one coefficient computed at node i

computational redundancy. The **cyclic reduction** algorithm is a modification of recursive doubling which reduces the amount of computations to be performed. In each step, half the variables in the equation system are eliminated which means that only half of the values $a_i^{(k)}$, $b_i^{(k)}$, $c_i^{(k)}$, and $y_i^{(k)}$ are computed. A substitution phase is needed to compute the solution vector x. The elimination and the substitution phases of cyclic reduction are described by the following two phases:

1. Elimination phase: For $k = 1, \ldots, \lfloor \log n \rfloor$ compute $a_i^{(k)}$, $b_i^{(k)}$, $c_i^{(k)}$, and $y_i^{(k)}$ with $i = 2^k, \ldots, n$ and step size 2^k. The number of equations of the form (7.26) is reduced by a factor of $1/2$ in each step. In step $k = \lfloor \log n \rfloor$ there is only one equation left for $i = 2^N$ with $N = \lfloor \log n \rfloor$.
2. Substitution phase: For $k = \lfloor \log n \rfloor, \ldots, 0$ compute x_i according to Eq. (7.26) for $i = 2^k, \ldots, n$ with step size 2^{k+1}:

$$x_i = \frac{y_i^{(k)} - a_i^{(k)} \cdot x_{i-2^k} - c_i^{(k)} \cdot x_{i+2^k}}{b_i^{(k)}} . \qquad (7.27)$$

Figure 7.11 illustrates the computations of the elimination and the substitution phases of cyclic reduction represented by nodes and their dependencies represented by arrows. In each computation step k, $k = 1, \ldots, \lfloor \log n \rfloor$, of the elimination phase,

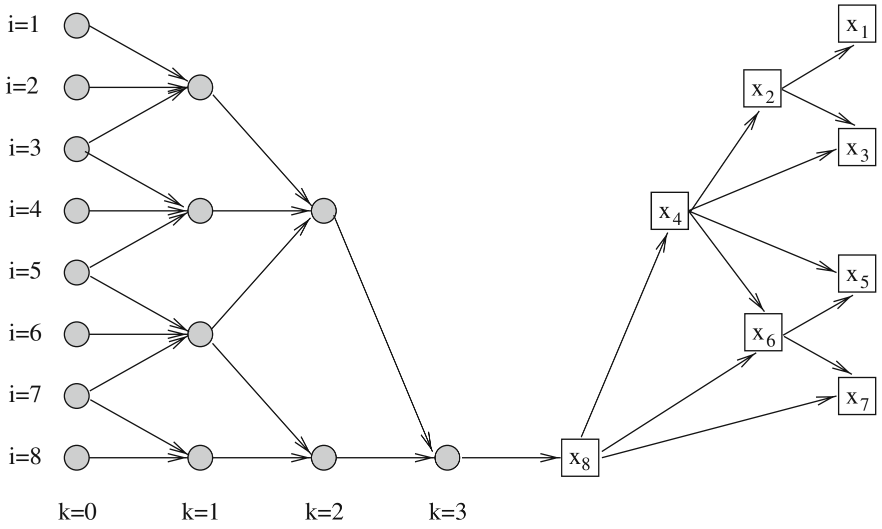

Fig. 7.11 Dependence graph illustrating the dependencies between neighboring computation steps of the cyclic reduction algorithm for the case of three computation steps and eight equations in analogy to the representation in Fig. 7.10. The first four columns represent the computations of the coefficients. The last columns in the graph represent the computation of the solution vector x in the second phase of the cyclic reduction algorithm, see (7.27)

there are $n/2^k$ nodes representing the computations for the coefficients of one equation. This results in

$$\frac{n}{2}+\frac{n}{4}+\frac{n}{8}+\cdots+\frac{n}{2^N}=n\cdot\sum_{i=1}^{\lfloor\log n\rfloor}\frac{1}{2^i}\le n$$

computation nodes with $N = \lfloor\log n\rfloor$ and, therefore, the execution time of cyclic reduction is $O(n)$. Thus, the computational complexity is the same as for the Gaussian elimination; however, the cyclic reduction offers potential parallelism which can be exploited in a parallel implementation as described in the following.

The computations of the numbers $\alpha_i^{(k)}$, $\beta_i^{(k)}$ require a division by $b_i^{(k)}$ and, thus, cyclic reduction as well as recursive doubling is not possible if any number $b_i^{(k)}$ is zero. This can happen even when the original matrix is invertible and has non-zero diagonal elements or when the Gaussian elimination can be applied without pivoting. However, for many classes of matrices it can be shown that a division by zero is never encountered. Examples are matrices A which are symmetric and positive definite or invertible and diagonally dominant, see [61] or [115] (using the name odd–even reduction). (A matrix A is symmetric if $A = A^T$ and positive definite if $x^T Ax > 0$ for all x. A matrix is diagonally dominant if in each row the absolute value of the diagonal element exceeds the sum of the absolute values of the other elements in the row without the diagonal in the row.)