ELEN 162
A. I. Zecevic

Lecture Notes for Week 10

Iterative Methods

Although this lecture will focus mainly on solving systems of linear equations, we will begin by considering a more general problem of the form

$$f(x) = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) \end{bmatrix} = 0 \tag{1}$$

where functions $f_i(x_1, x_2, \ldots, x_n)$ are nonlinear. In our previous discussion, we established that solving equations of this sort poses some significant challenges, both practical and theoretical. One of the most notable ones stems from the fact that it can be difficult (and sometimes impossible) to determine whether such a system has a solution, and whether this solution is unique. We will now consider a special class of functions known as *contraction mappings*, for which these questions can be answered precisely.

Contraction Mappings

Before we explain what a contraction mapping is, we should first observe that equation (1) can be equivalently expressed as

$$F(x) = x \tag{2}$$

where $F(x)$ is defined as

$$F(x) = f(x) + x \tag{3}$$

We will refer to any solution x^* of equation (2) as a *fixed point* of F, because function F maps this vector into itself (in other words, $F(x^*) = x^*$).

We now introduce the following definition.

Definition 1. Function F is said to be a *contraction mapping* on set $X \subset R^n$ if there exists a number α such that $0 < \alpha < 1$ and

$$\|F(x) - F(y)\| \leq \alpha \|x - y\| \tag{4}$$

for any $x, y \in X$. Parameter α is commonly referred to as the *modulus* of F.

Remark 1. Note that this definition doesn't specify the type of norm for which inequality (4) holds - the only thing that matters is that such a norm *exists*.

The following theorem shows that any contraction mapping has a *unique fixed point*, and that this fixed point can be computed iteratively.

Theorem 10.1. Let $F(x)$ be a contraction mapping with modulus α which is defined on a closed set $X \subset R^n$. Then, F has a unique fixed point $x^* \in X$, and the sequence

$$x(k+1) = F(x(k)) \tag{5}$$

converges to x^* for *any* initial approximation $x(0)$. Iterates $x(k)$ that are obtained using expression (5) satisfy

$$\|x(k) - x^*\| \leq \alpha^k \|x(0) - x^*\| \tag{6}$$

for any $k \geq 0$.

Linear Equations and Contraction Mappings

One of the most straightforward applications of Theorem 10.1 arises in the context of linear algebraic equations. With that in mind, we will now examine three methods that use contraction mappings to solve such systems iteratively (one of which is ideally suited for parallelization).

All of these methods utilize the general framework that we described previously, and define function $F(x)$ as

$$F(x) = Gx + \xi \tag{7}$$

where

$$G = I - Q^{-1}A \tag{8}$$

and

$$\xi = Q^{-1}b \tag{9}$$

If x^* is the solution of system

$$Ax = b \tag{10}$$

we know that it will satisfy

$$x^* = Gx^* + \xi \equiv F(x^*) \tag{11}$$

as well, and can therefore guarantee that x^* will be a fixed point of function F.

Since

$$\|F(x) - F(y)\| = \|Gx - Gy\| \leq \|G\| \, \|x - y\| \tag{12}$$

holds true for any choice of norm, $F(x)$ will obviously be a contraction if $\|G\| < 1$. Under such circumstances, iterative sequence

$$x(k+1) = Gx(k) + \xi \tag{13}$$

will converge to the fixed point of $F(x)$ for any choice of $x(0)$, and $\|G\|$ will play the role of parameter α.

Remark 2. It is important to note in this context that if $\|G\| < 1$ is not a necessary condition for the convergence of sequence (13). As we explained earlier, this process will also converge if $\rho(G) < 1$, which is a less restrictive requirement (since $\rho(G) \leq \|G\|$).

The solution methods that we will consider next are variants of the approach described in equations (7) - (9), which differ primarily in the way matrix Q is chosen. In discussing the properties of these methods, we will implicitly assume that A is a *symmetric positive definite* matrix of dimension $n \times n$. We can do so without any loss of generality, since system (10) can always be rewritten as

$$A^T Ax = A^T b \tag{14}$$

The Jacobi Method

Suppose that matrix Q is chosen as

$$Q = \begin{bmatrix} A_{11} & 0 & \cdots & 0 \\ 0 & A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{pp} \end{bmatrix} \tag{15}$$

where A_{ii} $(i = 1, 2, \ldots, p)$ represent diagonal blocks of matrix A. System (10) can then be solved iteratively as

$$x(k+1) = Gx(k) + \xi \tag{16}$$

where

$$G = I - Q^{-1}A \tag{17}$$

If we multiply both sides of (16) by Q, this becomes

$$Qx(k+1) = (Q - A)x(k) + b \tag{18}$$

(since $\xi = Q^{-1}b$ by definition).

The fact that Q consists of the diagonal blocks of A allows us to parallelize this process in a straighforward manner. Indeed, since

$$Q - A = \begin{bmatrix} 0 & -A_{12} & \cdots & -A_{1p} \\ -A_{21} & 0 & \cdots & -A_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ -A_{p1} & -A_{p2} & \cdots & 0 \end{bmatrix} \tag{19}$$

each processor can compute $x_i(k+1)$ *independently*, by solving system

$$A_{ii}x_i(k+1) = -\sum_{j \neq i} A_{ij}x_j(k) + b_i \tag{20}$$

(this can be done easily using LU factorization or Gaussian elimination). It is assumed that vectors $x_j(k)$ $(j \neq i)$ are available to processor i at this point, since each iteration is followed by a multinode broadcast (in which the updated values are exchanged).

An important advantage of this method is the fact that we can choose the dimensions of the diagonal blocks freely. As a result, the problem can be easily adapted to accomodate any number of processors. This is particularly useful in cases when A is *not* a sparse matrix, because it allows us to reduce the sizes of blocks A_{ii} (which need to be factorized by individual processors). In the extreme case when we have n processors at our disposal, matrix

$$Q = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \tag{21}$$

becomes purely diagonal, and we can compute $x_i(k+1)$ directly as

$$x_i(k+1) = -\frac{1}{a_{ii}} \sum_{j \neq i} a_{ij} x_j(k) + \frac{1}{a_{ii}} b_i \tag{22}$$

When the iterative process has the special form shown in (22), there is a simple condition that ensures its convergence. To see what this condition is, let us first observe that the corresponding matrix G will have the form

$$G = I - Q^{-1}A = \begin{bmatrix} 0 & -a_{12}/a_{11} & \cdots & -a_{1n}/a_{11} \\ -a_{21}/a_{22} & 0 & \cdots & -a_{2n}/a_{22} \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n1}/a_{nn} & -a_{n2}/a_{nn} & \cdots & 0 \end{bmatrix} \tag{23}$$

If we assume that

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \quad (i = 1, 2, \ldots, n) \tag{24}$$

(which is a property known as *diagonal dominance*) it follows that

$$\frac{1}{|a_{ii}|} \sum_{j \neq i} |a_{ij}| < 1 \quad (i = 1, 2, \ldots, n) \tag{25}$$

and norm $\|G\|_\infty$ can be bounded as

$$\|G\|_\infty = \max_i \sum_{j=1}^{n} |g_{ij}| = \max_i \frac{1}{|a_{ii}|} \sum_{j \neq i} |a_{ij}| < 1 \tag{26}$$

This ensures that $F(x) = Gx + \xi$ is a contraction, whose modulus is $\alpha = \|G\|_\infty$.

When discussing the convergence properties of the Jacobi method, we should also mention that sequence (16) is always *symmetrizable*, since matrix A is assumed to be symmetric and positive definite. This means that convergence can be achieved even if $\rho(G) \geq 1$, provided that equation (16) is appropriately modified. What makes this possible is the fact that the diagonal blocks of a symmetric positive definite matrix have this property as well (and so does matrix Q).

As we previously established, under such circumstances the iterative sequence

$$x(k+1) = G_\gamma x(k) + \gamma \xi \tag{27}$$

will converge to the solution of (10) if we set

$$G_\gamma = \gamma G + (1 - \gamma) I \tag{28}$$

and choose parameter γ as

$$\gamma = \frac{2}{2 - \lambda_m(G) - \lambda_M(G)} \tag{29}$$

The following example illustrates how the Jacobi methods work when it is modified in this way.

Example 1. Let us consider the system

$$\begin{bmatrix} 1 & 0 & 3 \\ 2 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \tag{30}$$

Since matrix A is neither symmetric nor positive definite in this case, our first step will be to multiply both sides of equation (30) by A^T. When we do so, we obtain

$$\begin{bmatrix} 6 & 5 & 6 \\ 5 & 5 & 3 \\ 6 & 3 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix} \tag{31}$$

In order to apply the Jacobi method to system (31), we will use a diagonal matrix of the form

$$Q = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 11 \end{bmatrix} \tag{32}$$

and compute matrix G and vector ξ as

$$G = I - Q^{-1}A = \begin{bmatrix} 0 & -0.833 & -1 \\ -1 & 0 & -0.6 \\ -0.545 & -0.273 & 0 \end{bmatrix} \tag{33}$$

and

$$\xi = Q^{-1}b = \begin{bmatrix} 0.333 \\ 0.2 \\ 0.364 \end{bmatrix} \tag{34}$$

The iterative process will then assume the usual form

$$x(k+1) = Gx(k) + \xi \tag{35}$$

and we can execute it in parallel using 3 processors if the convergence conditions are met.

The difficulty in this case is that $\|G\|>1$ for *any* choice of norm, so we cannot guarantee that $F(x) = Gx + \xi$ is a contraction mapping. Condition $\rho(G) < 1$ isn't satisfied either, since the spectral radius of matrix G is $\rho(G) = 1.39$. We must therefore modify the iterative sequence as

$$x(k+1) = G_\gamma x(k) + \omega \tag{36}$$

where

$$G_\gamma = \gamma G + (1-\gamma)I \tag{37}$$

and

$$\omega = \gamma\xi \tag{38}$$

Setting

$$\gamma = \frac{2}{2 - \lambda_m(G) - \lambda_M(G)} = 0.835778 \tag{39}$$

we obtain matrix

$$G_\gamma = \begin{bmatrix} 0.1642 & -0.6965 & -0.8358 \\ -0.8358 & 0.1642 & -0.5015 \\ -0.4559 & -0.2280 & 0.1642 \end{bmatrix} \tag{40}$$

whose spectral radius is $\rho(G_\gamma) = 0.9983$.

The fact that $\rho(G_\gamma)$ is very close to 1 suggests that sequence (36) might converge very slowly. To show that this is indeed the case, let us consider what happens if we set $x(0) = [0 \;\; 0 \;\; 0]^T$ as the starting point of the iterative process. With this choice of $x(0)$, it takes about 4,000 iterations to produce vector

$$\bar{x} = \begin{bmatrix} -4.9954 \\ 3.9961 \\ 1.9983 \end{bmatrix} \tag{41}$$

which approximates the exact solution

$$x^* = \begin{bmatrix} -5 \\ 4 \\ 2 \end{bmatrix} \tag{42}$$

with an error of

$$\|\bar{x} - x^*\|_\infty = 0.0046 \tag{43}$$

If we choose $x(0) = [-4.5 \;\; 3.5 \;\; 2]^T$ (which is much closer to x^*), the situation improves, but not dramatically. With this initial approximation, we obtain

$$\bar{x} = \begin{bmatrix} -4.9621 \\ 3.9746 \\ 1.9903 \end{bmatrix} \tag{44}$$

after 1,500 iterations.

What this example tells us is that the Jacobi method needn't always be effective. Whether or not this will be the case depends to a large extent on matrix A, and how the diagonal blocks that define matrix Q are chosen. If the convergence rate happens to be slow (which is a distinct possibility), this can easily offset the potential benefits of parallelization.

The Gauss-Seidel Method

In the Gauss-Seidel method, matrix Q is assumed to have the form

$$Q = \begin{bmatrix} A_{11} & 0 & \cdots & 0 \\ A_{21} & A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & A_{pp} \end{bmatrix} \tag{45}$$

To analyze the properties of matrix

$$G = I - Q^{-1}A \tag{46}$$

when Q is chosen in this manner, it will be convenient to define auxiliary matrices D, C_L and C_U as

$$D = \begin{bmatrix} A_{11} & 0 & 0 & \cdots & 0 \\ 0 & A_{22} & 0 & \cdots & 0 \\ 0 & 0 & A_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A_{pp} \end{bmatrix} \tag{47}$$

$$C_L = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ -A_{21} & 0 & 0 & \cdots & 0 \\ -A_{31} & -A_{32} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -A_{p1} & -A_{p2} & -A_{p3} & \cdots & 0 \end{bmatrix} \tag{48}$$

and

$$C_U = \begin{bmatrix} 0 & -A_{12} & -A_{13} & \cdots & -A_{1p} \\ 0 & 0 & -A_{23} & \cdots & -A_{2p} \\ 0 & 0 & 0 & \cdots & -A_{3p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{49}$$

We can then express matrices A and Q as

$$A = D - C_L - C_U \tag{50}$$

and

$$Q = D - C_L \tag{51}$$

respectively.

Equations (50) and (51) allow us to rewrite (46) as

$$G = I - (D - C_L)^{-1}(D - C_L - C_U) = (D - C_L)^{-1}C_U \tag{52}$$

If we do so, the iterative process will have the form

$$x(k+1) = (D - C_L)^{-1}C_U x(k) + \xi \tag{53}$$

where

$$\xi = Q^{-1}b = (D - C_L)^{-1}b \tag{54}$$

Multiplying both sides of (53) by $D - C_L$, this becomes

$$(D - C_L)x(k+1) = C_U x(k) + b \tag{55}$$

which is equivalent to

$$Dx(k+1) = C_L x(k+1) + C_U x(k) + b \tag{56}$$

It is important to recognize at this point that the form of the Gauss-Seidel iterative sequence is *different* from the one that corresponds to the Jacobi method, since elements of

vector $x(k+1)$ appear on *both* sides of expression (56). To see what this means in practical terms, let us rewrite (56) as

$$A_{ii}x_i(k+1) = -\sum_{j=1}^{i-1} A_{ij}x_j(k+1) - \sum_{j=i+1}^{p} A_{ij}x_j(k) + b_i \tag{57}$$

(using the definitions of D, C_L and C_U). From (57) it is readily observed that processor i *cannot compute* $x_i(k+1)$ *until it receives updated information from processors* $1, \ldots, i-1$. This indicates that the Gauss-Seidel method is difficult to parallelize unless some additional assumptions are made about the structure of matrix A.

We will return to the question of parallelization shortly, but before we do that, we first need to consider the convergence properties of sequence (53). The first thing to notice in this context is that Q is *not* a symmetric matrix, which means that the Gauss-Seidel iterative process needn't necessarily be symmetrizable. As a result, we cannot guarantee that replacing G with

$$G_\gamma = \gamma G + (1-\gamma)I \tag{58}$$

will result in a convergent sequence in cases when $\rho(G) \geq 1$. The following theorem demonstrates, however, that this will not be an issue if matrix A is symmetric and positive definite.

Theorem 10.4. Suppose that matrix A is symmetric and positive definite. In that case, the spectral radius of matrix

$$G = (D - C_L)^{-1}C_U \tag{59}$$

always satisfies $\rho(G) < 1$.

The following example illustrates how the Gauss-Seidel method operates, and provides some useful insights about its convergence properties.

Example 2. Let us once again consider the system

$$\begin{bmatrix} 1 & 0 & 3 \\ 2 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \tag{60}$$

which we already examined in Example 1. Since matrix A is neither symmetric nor positive definite in this case, we will have to multiply both sides of (60) by A^T, which produces

$$\begin{bmatrix} 6 & 5 & 6 \\ 5 & 5 & 3 \\ 6 & 3 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix} \tag{61}$$

Given that matrix A^TA is symmetric and positive definite by construction, the iterative sequence is guaranteed to converge if we set

$$Q = \begin{bmatrix} 6 & 0 & 0 \\ 5 & 5 & 0 \\ 6 & 3 & 11 \end{bmatrix} \tag{62}$$

and compute matrix G and vector ξ as

$$G = I - Q^{-1}A = \begin{bmatrix} 0 & -0.833 & -1 \\ 0 & 0.833 & 0.4 \\ 0 & 0.228 & 0.436 \end{bmatrix} \tag{63}$$

and

$$\xi = Q^{-1}b = \begin{bmatrix} 0.333 \\ -0.133 \\ 0.210 \end{bmatrix} \tag{64}$$

If we choose $x(0) = [0 \;\; 0 \;\; 0]^T$ as our starting point (as we did with the Jacobi method), the iterative process

$$x(k+1) = Gx(k) + \xi \tag{65}$$

produces

$$\bar{x} = \begin{bmatrix} -4.9957 \\ 3.9966 \\ 1.9986 \end{bmatrix} \tag{66}$$

after 1,700 iterations. This vector approximates x^* with an error

$$\|\bar{x} - x^*\|_\infty = 0.0043 \tag{67}$$

which is very similar to the value obtained in (43). Using $x(0) = [-4.5 \;\; 3.5 \;\; 2]^T$ as the initial approximation, we get

$$\bar{x} = \begin{bmatrix} -4.9682 \\ 3.9744 \\ 1.9896 \end{bmatrix} \tag{68}$$

after 650 iterations.

If we compare Examples 1 and 2, it is readily observed that the Gauss-Seidel process converges significantly faster than the Jacobi method (roughly by a factor of 2). It is important to recognize, however, that the number of iterations is still high, which could pose a significant problem in cases when the matrix is large. We therefore need to examine whether slow convergence is an inherent property of the Jacobi and Gauss-Seidel methods. The following example shows that this isn't the case, and that there are situations when both techniques require only a modest number of iterations.

Example 3. Let us consider the system

$$\begin{bmatrix} 5 & -2 & 3 \\ -2 & 9 & 1 \\ 3 & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \tag{69}$$

whose exact solution is

$$x^* = \begin{bmatrix} 0.53271 \\ 0.27103 \\ -0.37383 \end{bmatrix} \tag{70}$$

It is easily verified that matrix A is symmetric and positive definite, so equation (69) doesn't need to be modified in any way.

Setting

$$Q = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 5 \end{bmatrix} \tag{71}$$

we obtain

$$G = I - Q^{-1}A = \begin{bmatrix} 0 & 0.4 & -0.6 \\ 0.222 & 0 & -0.111 \\ -0.6 & -0.2 & 0 \end{bmatrix} \tag{72}$$

and

$$\xi = Q^{-1}b = \begin{bmatrix} 0.2 \\ 0.111 \\ 0 \end{bmatrix} \tag{73}$$

respectively. Since $\rho(G) = 0.7372$ in this case, the Jacobi iterative sequence is guaranteed to converge for any choice of $x(0)$. Starting from $x(0) = [0 \;\; 0 \;\; 0]^T$, we obtain

$$\bar{x} = \begin{bmatrix} 0.53271 \\ 0.27092 \\ -0.37361 \end{bmatrix} \tag{74}$$

after 25 iterations. It is not difficult to see that this approximation is quite accurate, since

$$\|\bar{x} - x^*\|_\infty = 0.00024 \tag{75}$$

In order to apply the Gauss-Seidel method to this problem, we will choose matrix Q as

$$Q = \begin{bmatrix} 5 & 0 & 0 \\ -2 & 9 & 0 \\ 3 & 1 & 5 \end{bmatrix} \tag{76}$$

in which case we obtain

$$G = I - Q^{-1}A = \begin{bmatrix} 0 & 0.4 & -0.6 \\ 0 & 0.089 & -0.244 \\ -0.6 & -0.258 & 0.409 \end{bmatrix} \tag{77}$$

and

$$\xi = Q^{-1}b = \begin{bmatrix} 0.2 \\ 0.156 \\ -0.151 \end{bmatrix} \tag{78}$$

The fact that A is symmetric and positive definite guarantees convergence, and the Gauss-Seidel method produces

$$\bar{x} = \begin{bmatrix} 0.53264 \\ 0.27100 \\ -0.37378 \end{bmatrix} \tag{79}$$

after only 15 iterations (starting from $x(0) = [0 \;\; 0 \;\; 0]^T$).

Example 3 confirms our previous observation that the Gauss-Seidel method typically converges about twice as fast as the Jacobi method. This is clearly an advantage, but it is offset

by the fact that the Gauss-Seidel process is *inherently sequential*, and is therefore difficult to parallelize. There is a class of problems, however, where these shortcomings can be largely avoided by exploiting the structure of matrix A. Systems of this sort arise frequently in the numerical solution of partial differential equations, where certain discretization techniques are known to produce matrices with very regular nonzero patterns. The following example illustrates how this property can be used to effectively parallelize Gauss-Seidel iterations.

Example 4. Let us consider a symmetric matrix of dimension 16×16, whose nonzero pattern is shown in (80). The graph that corresponds to this matrix has a grid-like structure, and is displayed in Fig. 1.

$$A = \begin{bmatrix}
* & * & & & * & & & & & & & & & & & \\
* & * & * & & & * & & & & & & & & & & \\
 & * & * & * & & & * & & & & & & & & & \\
 & & * & * & & & & * & & & & & & & & \\
* & & & & * & * & & & * & & & & & & & \\
 & * & & & * & * & * & & & * & & & & & & \\
 & & * & & & * & * & * & & & * & & & & & \\
 & & & * & & & * & * & & & & * & & & & \\
 & & & & * & & & & * & * & & & * & & & \\
 & & & & & * & & & * & * & * & & & * & & \\
 & & & & & & * & & & * & * & * & & & * & \\
 & & & & & & & * & & & * & * & & & & * \\
 & & & & & & & & * & & & & * & * & & \\
 & & & & & & & & & * & & & * & * & * & \\
 & & & & & & & & & & * & & & * & * & * \\
 & & & & & & & & & & & * & & & * & *
\end{bmatrix} \tag{80}$$

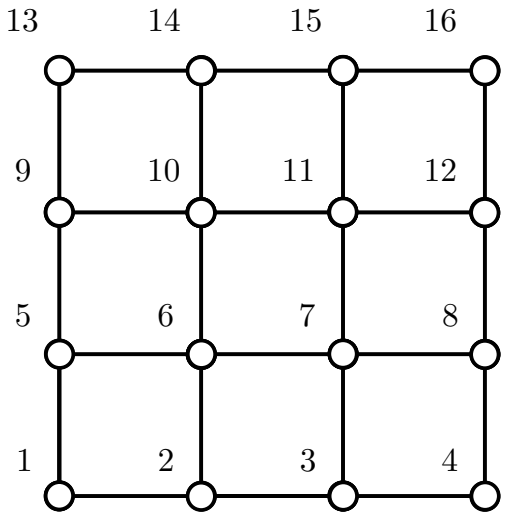

Figure 1: The graph that corresponds to matrix (80).

In order to modify this matrix in a way that is suitable for Gauss-Seidel iterations, let us temporarily label the nodes of the graph using *pairs* of indices (in the manner shown in Fig. 2). We will then group the nodes into two different sets - *black* nodes (which correspond to indices (i, j) where $i + j$ is odd) and *white* nodes (where $i + j$ is even). When we do so, we obtain the graph shown in Fig. 3, in which each white node has only black neighbors and vice versa.

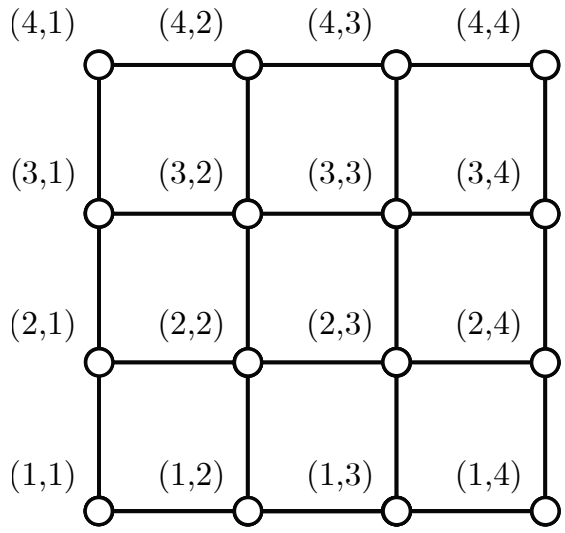

Figure 2: A different labeling of the nodes.

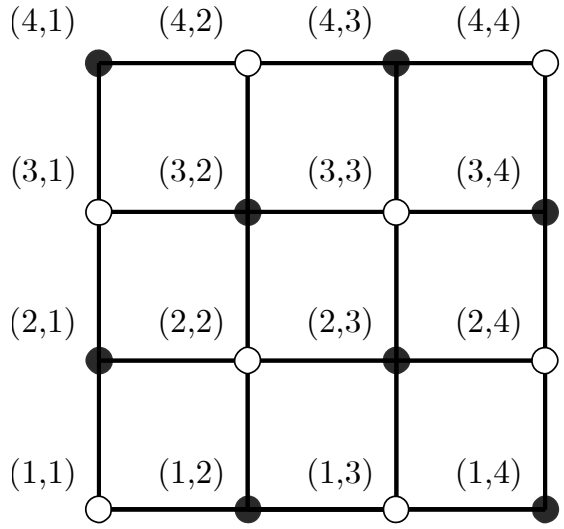

Figure 3: A black-white partitioning of the nodes.

The rationale behind this strategy is based on the observation that the neighbors of node (i, j) have indices $\{(i, j-1), (i, j+1), (i-1, j)(i+1, j)\}$. Consequently, if (i, j) happens to

be a white node (which means that $i+j$ is an even number), all of its neigbors will be *black*, since the sum of their indices is either $i+j-1$ or $i+j+1$. The same would obviously be true if (i,j) were a black node.

The advantages of such a partitioning become apparent if we label white nodes first, followed by black nodes. The resulting graph (which is shown in Fig. 4) represents a renumbered version of the graph in Fig. 1, which corresponds to permutation vector

$$p = \begin{bmatrix} 1 & 3 & 6 & 8 & 9 & 11 & 14 & 16 & 2 & 4 & 5 & 7 & 10 & 12 & 13 & 15 \end{bmatrix} \tag{81}$$

When we apply this permutation to the rows and columns of matrix A, we obtain the matrix shown in (82).

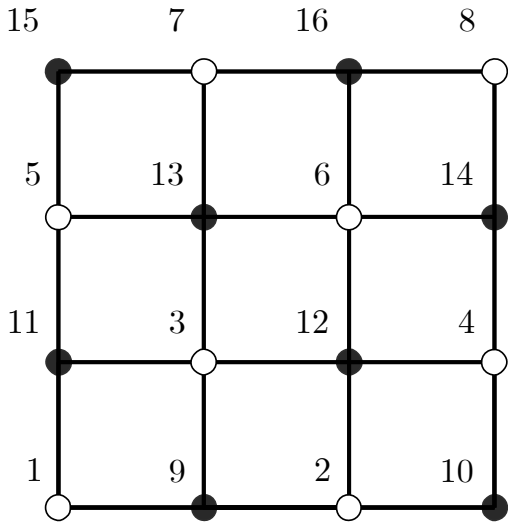

Figure 4: The renumbered graph.

$$\tilde{A} = \left[\begin{array}{cccccccc|cccccccc}
* & & & & & & & & * & & * & & & & & \\
 & * & & & & & & & * & * & & * & & & & \\
 & & * & & & & & & * & & * & * & * & & & \\
 & & & * & & & & & & * & & * & & * & & \\
 & & & & * & & & & & & * & & * & & * & \\
 & & & & & * & & & & & & * & * & * & & * \\
 & & & & & & * & & & & & & * & & * & * \\
 & & & & & & & * & & & & & & * & & * \\
\hline
* & * & * & & & & & & * & & & & & & & \\
 & * & & * & & & & & & * & & & & & & \\
* & & * & & * & & & & & & * & & & & & \\
 & * & * & * & & * & & & & & & * & & & & \\
 & & * & & * & * & * & & & & & & * & & & \\
 & & & * & & * & & * & & & & & & * & & \\
 & & & & * & & * & & & & & & & & * & \\
 & & & & & * & * & * & & & & & & & & *
\end{array}\right] \tag{82}$$

Let us now assume that we have 16 processors at our disposal, and that we wish to parallelize the Gauss-Seidel iterations. The components of vector x that are associated with white nodes can obviously be updated in parallel, since the partitioning ensures that $\tilde{a}_{ij} = 0$ for $i, j \leq 8$ (except for the diagonal elements, of course). As a result, processors whose task is to update these components can compute $x_i(k+1)$ as

$$x_i(k+1) = -\frac{1}{\tilde{a}_{ii}} \sum_{j=9}^{16} \tilde{a}_{ij} x_j(k) + b_i \quad (i = 1, 2, \ldots, 8) \tag{83}$$

Once this step is completed, the processors that are associated with white nodes can send their updated values to all other processors. At this point, the processors that correspond to black nodes can compute

$$x_i(k+1) = -\frac{1}{\tilde{a}_{ii}} \sum_{j=1}^{8} \tilde{a}_{ij} x_j(k+1) + b_i \quad (i = 9, 10, \ldots, 16) \tag{84}$$

in parallel, since components $\{x_j(k+1)\}$ $(j = 1, 2, \ldots, 8)$ are now available.

This example demonstrates that the Gauss-Seidel method can be parallelized efficiently if matrix A allows for a black-white partitioning. In such cases, only two sequential steps are required, and all other computations can be performed simultaneously. We should reiterate, however, that this approach works only for problems where the nonzero pattern of matrix A has a special structure. In a more general scenario, Gauss-Seidel iterations will typically have an irreducible sequential component, and will therefore be difficult to parallelized efficiently.

The SOR Method

Successive overrelaxation (SOR) can be viewed as a generalization of the Gauss-Seidel method, which allows for faster convergence when the parameter ω that defines it is optimally chosen. Since the values for this parameter are typically larger than 1, we use the term "overrelaxation" to characterize such iterative processes.

In this approach, matrix Q is chosen as

$$Q = \begin{bmatrix} \omega^{-1} A_{11} & 0 & \cdots & 0 \\ A_{21} & \omega^{-1} A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & \omega^{-1} A_{pp} \end{bmatrix} \tag{85}$$

which is equivalent to

$$Q = \frac{1}{\omega} D - C_L \tag{86}$$

It is not difficult to see that such a choice includes the Gauss-Siedel method as a special case (which corresponds to $\omega = 1$).

Recalling that

$$A = D - C_L - C_U \tag{87}$$

matrix G can be expressed as

$$G = I - Q^{-1} A = I - (\frac{1}{\omega} D - C_L)^{-1} (D - C_L - C_U) \tag{88}$$

and the iterative sequence becomes

$$x(k+1) = \left[I - (\frac{1}{\omega}D - C_L)^{-1}(D - C_L - C_U) \right] x(k) + (\frac{1}{\omega}D - C_L)^{-1} b \tag{89}$$

Multiplying both sides of (89) by $(\omega^{-1}D - C_L)$, we obtain

$$(\frac{1}{\omega}D - C_L)x(k+1) = \left[(\frac{1}{\omega}D - C_L) - (D - C_L - C_U) \right] x(k) + b \tag{90}$$

which is more convenient from a computational perspective.

To see how the workload can be distributed across a set of p processors, let us first multiply both sides of (90) by ω, which produces

$$(D - \omega C_L)x(k+1) = \left[(D - \omega C_L) - \omega(D - C_L - C_U) \right] x(k) + \omega b \tag{91}$$

Since this expression can be rewritten as

$$\begin{aligned} &Dx(k+1) = \omega C_L x(k+1) + \left[(D - \omega C_L) - \omega(D - C_L - C_U) \right] x(k) + \omega b = \\ &= \omega C_L x(k+1) + \left[(1-\omega)D + \omega C_U \right] x(k) + \omega b \end{aligned} \tag{92}$$

we can conclude that processor i can compute $x_i(k+1)$ as

$$\begin{aligned} &A_{ii}x_i(k+1) = -\omega \sum_{j=1}^{i-1} A_{ij}x_j(k+1) + (1-\omega)A_{ii}x_i(k) - \\ &-\omega \sum_{j=i+1}^{p} A_{ij}x_j(k) + \omega b_i \end{aligned} \tag{93}$$

The convergence analysis for the SOR method is fairly complicated, so we will not discuss it in detail. For our purposes, it suffices to say that sequence (93) will converge if ω satisfies $0 < \omega < 2$ (provided that A is a symmetric positive definite matrix). The challenge then becomes finding the value of ω that maximizes the convergence rate. This is not a trivial problem, but once such an ω is identified, the SOR method generally becomes significantly faster than the other two techniques that we discussed so far. When it comes to parallel computing, however, sequence (93) faces the same issues as the Gauss-Seidel iterative method. It is therefore fair to say that this approach can be effectively parallelized only for certain special classes of problems.

The Conjugate Gradient Method

The conjugate gradient method provides an alternative approach for solving systems of linear equations which does not rely on contraction mappings. When analyzing this approach, it is standard practice to assume that matrix A is symmetric and positive definite. As we already noted, this is in no way restrictive, since system (10) can always be rewritten as

$$A^T A x = A^T b \tag{94}$$

In the following, we will additionally assume that $b = 0$, since that allows us to present the main features of the algorithm in the simplest possible way. At the end of our discussion we will consider the general scenario when $b \neq 0$, and show that it requires only minor modifications to the procedure that we will now describe.

The conjugate gradient method makes use of the fact that minimizing the quadratic form

$$F(x) = \frac{1}{2}x^T Ax \tag{95}$$

is equivalent to solving the system

$$Ax = 0 \tag{96}$$

Although this result seems intuitively obvious, it will be helpful to explain the connection between expressions (95) and (96) a bit more precisely. In order to do that, we should first note that vector x^* minimizes $F(x)$ if

$$F(x^* + \Delta x) - F(x^*) > 0 \tag{97}$$

for any $\Delta x \neq 0$. Observing that

$$\begin{aligned} &F(x + \Delta x) - F(x) = \frac{1}{2}(x + \Delta x)^T A(x + \Delta x) - \frac{1}{2}x^T Ax = \\ &= \frac{1}{2}x^T A\Delta x + \frac{1}{2}\Delta x^T Ax + \Delta x^T A\Delta x = x^T A\Delta x + \Delta x^T A\Delta x \end{aligned} \tag{98}$$

the term that multiplies Δx can be identified as the Jacobian of function F. In applications related to optimization, the transpose of this Jacobian is commonly referred to as the *gradient of* F (and is denoted $\nabla F(x)$). In view of that, we can rewrite expression (98) as

$$F(x + \Delta x) - F(x) = [\nabla F(x)]^T \Delta x + \Delta x^T A\Delta x \tag{99}$$

Recalling that A is a positive definite matrix, it is not difficult to show that expression (99) will be positive for any choice of Δx if and only if

$$\nabla F(x) = 0 \tag{100}$$

Since $\nabla F(x)$ equals Ax, equations (96) and (100) are equivalent, and therefore have the same solution x^*. This allows us to treat systems of linear algebraic equations as optimization problems, and apply appropriate minimization techniques.

One of these techniques is the conjugate gradient method, which computes x^* by producing a sequence of the form

$$x(k+1) = x(k) + \rho(k)y(k) \tag{101}$$

In this sequence, scalar $\rho(k)$ and vector $y(k)$ are chosen in such a way that

$$F(x(k) + \rho(k)y(k)) < F(x(k) + \beta y(k)) \tag{102}$$

holds for any real number $\beta \neq \rho(k)$. What this means is that $\rho(k)$ *minimizes function* F *when we move from* $x(k)$ *along the direction defined by* $y(k)$.

If $y(k) \neq 0$, it is always possible to find a number $\rho(k)$ that satisfies condition (102). To see how this can be done, let us introduce an auxliliary function $\Phi(\beta) = F(x(k) + \beta y(k))$, which depends only on β when vectors $x(k)$ and $y(k)$ are fixed. It is easily verified that $\Phi(\beta)$ can be expressed as

$$\begin{aligned} \Phi(\beta) = F(x(k) + \beta y(k)) = F(x(k)) + \\ + [\nabla F(x(k))]^T \beta y(k) + \frac{1}{2}\beta^2 y(k)^T A y(k) \end{aligned} \tag{103}$$

(this follows directly from equation (99), by setting $x = x(k)$ and $\Delta x = \beta y(k)$).

In order to minimize this function, we need to ensure that

$$\frac{d\Phi}{d\beta} = y^T(k)[\nabla F(x(k))] + \beta y(k)^T A y(k) = 0 \tag{104}$$

This will obviously be the case if

$$\beta = -\frac{y^T(k)[\nabla F(x(k))]}{y(k)^T A y(k)} \tag{105}$$

which tells us that we should use expression (105) to compute $\rho(k)$ once we have vector $y(k)$.

To see how $y(k)$ should be chosen, we will need the following definition.

Definition 2. Two vectors x and y are said to be *A-conjugate* if they satisfy

$$x^T A y = 0 \tag{106}$$

This definition is useful because the iterative process described in (101) requires vectors $\{y(k)\}$ $(k = 0, 1, \ldots)$ to be mutually A-conjugate. We will describe how such vectors can be computed shortly, but for now it will suffice to recognize that this is always possible.

The following theorem describes three important properties of sequence (101) if vectors $\{y(k)\}$ satisfy

$$y^T(i) A y(k) = 0 \tag{107}$$

when $i \neq k$, and $\rho(k)$ is chosen in the manner indicated in (105). To simplify the notation, in this theorem (and in all subsequent derivations) we will denote $\nabla F(x(k))$ by $g(k)$.

Theorem 10.5. Let $\{y(0), y(1), \ldots, y(k)\}$ be a set of A-conjugate vectors, and let $x(k)$ be the k-th iterate produced by the conjugate gradient method using these vectors. If $\rho(k)$ is chosen as

$$\rho(k) = -\frac{y^T(k) g(k)}{y(k)^T A y(k)} \tag{108}$$

in each step, we can guarantee that:

(a) Vectors $\{y(0), y(1), \ldots, y(k)\}$ will be linearly independent.

(b) The gradient of $F(x)$ at $x = x(k+1)$ satisfies

$$g^T(k+1) y(i) = 0 \quad (i = 0, 1, \ldots, k) \tag{109}$$

(c) The sequence of vectors $\{x(k)\}$ obtained using (101) satisfies

$$F(x(k+1)) < F(x(k)) < \ldots < F(x(0)) \tag{110}$$

Theorem 10.5 is important because it ensures that $F(x)$ becomes smaller in each step if $\rho(k)$ is chosen in the manner indicated in (108), and if vectors $\{y(k)\}$ are mutually A-conjugate. It also allows us to compute these vectors *recursively*, by setting

$$y(0) = -g(0) \tag{111}$$

and computing each subsequent vector $y(k)$ as

$$y(k) = -g(k) + \sigma_{k-1}(k)y(k-1) \tag{112}$$

The coefficient $\sigma_{k-1}(k)$ that appears in expression (112) is defined as

$$\sigma_{k-1}(k) = \frac{g^T(k)g(k)}{g^T(k-1)g(k-1)} \tag{113}$$

and can be easily calculated if vectors $x(k)$ and $x(k-1)$ are known.

Remark 3. A more detailed explanation of this procedure can be found in the textbook (including a proof which demonstrates that vectors $y(k)$ obtained in this manner are mutually A-conjugate).

Now that we have a general idea of how the conjugate gradient method works, we need to consider how it can be generalized to systems of the form

$$Ax = b \tag{114}$$

It turns out that this requires only a minor modification of the process, which amounts to recognizing that $g(k)$ becomes

$$g(k) = Ax(k) - b \tag{115}$$

when $b \neq 0$.

To see why this is so, let us express the solution of equation (114) as $x^* = A^{-1}b$. Since A is a symmetric positive definite matrix by assumption, we know that function

$$F(x) = \frac{1}{2}(x - x^*)^T A(x - x^*) \tag{116}$$

will have a global minimum for $x = x^*$. It is not difficult to show that

$$\nabla F(x) = Ax - b \tag{117}$$

in this case, which implies that equation (114) is equivalent to

$$\nabla F(x) = 0 \tag{118}$$

We can therefore solve this problem using the same procedure as before, the only difference being that we now have

$$g(k) = \nabla F(x(k)) = Ax(k) - b \tag{119}$$

instead of

$$g(k) = Ax(k) \tag{120}$$

The convergence conditions for the conjugate gradient method are quite straightforward, because they do not depend on matrix A (or on the choice of initial approximation). Unlike the Jacobi and Gauss-Sedel methods, in this case the iterative sequence always produces the *exact* solution, and there is an upper bound on the number of steps that are needed to accomplish this. The following lemma specifies what this bound looks like.

Lemma 10.5. If A is a symmetric positive definite matrix of dimension $n \times n$, the conjugate gradient method converges to the exact solution of system (10) in at most n steps.

We now provide an example that illustrates how the conjugate gradient method works when $b \neq 0$ (which is the most general scenario).

Example 4. Let us consider the system

$$\begin{bmatrix} 6 & 5 & 6 \\ 5 & 5 & 3 \\ 6 & 3 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix} \tag{121}$$

which we already analyzed in Example 1 (the exact solution of this system is given in (42)). Since matrix A is symmetric and positive definite, we don't need to make any modifications to (121), and can proceed directly with the iterations.

We will describe how this process evolves in a step by step manner, assuming that the initial approximation is chosen as $x(0) = [0 \;\; 0 \;\; 0]^T$.

Iteration 1

STEP 1

$$g(0) = Ax(0) - b = \begin{bmatrix} -2 \\ -1 \\ -4 \end{bmatrix} \tag{122}$$

STEP 2

$$y(0) = -g(0) = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix} \tag{123}$$

STEP 3

$$\rho(0) = -\frac{y^T(0)g(0)}{y^T(0)Ay(0)} = 0.06087 \tag{124}$$

STEP 4

$$x(1) = x(0) + \rho(0)y(0) = \begin{bmatrix} 0.121739 \\ 0.060870 \\ 0.243478 \end{bmatrix} \tag{125}$$

Iteration 2

STEP 1

$$g(1) = Ax(1) - b = \begin{bmatrix} 0.49565 \\ 0.64348 \\ -0.40870 \end{bmatrix} \tag{126}$$

STEP 2

$$\sigma_0(1) = \frac{g^T(1)g(1)}{g^T(0)g(0)} = 0.03937 \tag{127}$$

STEP 3

$$y(1) = -g(1) + \sigma_0(1)y(0) = \begin{bmatrix} -0.41691 \\ -0.60411 \\ 0.56618 \end{bmatrix} \tag{128}$$

STEP 4

$$\rho(1) = -\frac{y^T(1)g(1)}{y^T(1)Ay(1)} = 0.20528 \tag{129}$$

STEP 5

$$x(2) = x(1) + \rho(1)y(1) = \begin{bmatrix} 0.036157 \\ -0.063139 \\ 0.359700 \end{bmatrix} \tag{130}$$

Iteration 3

STEP 1

$$g(2) = Ax(2) - b = \begin{bmatrix} 0.059449 \\ -0.055809 \\ -0.015772 \end{bmatrix} \tag{131}$$

STEP 2

$$\sigma_1(2) = \frac{g^T(2)g(2)}{g^T(1)g(1)} = 0.0083428 \tag{132}$$

STEP 3

$$y(2) = -g(2) + \sigma_1(2)y(1) = \begin{bmatrix} -0.062927 \\ 0.050769 \\ 0.020496 \end{bmatrix} \tag{133}$$

STEP 4

$$\rho(2) = -\frac{y^T(2)g(2)}{y^T(2)Ay(2)} = 80.032 \tag{134}$$

STEP 5

$$x(3) = x(2) + \rho(2)y(2) = \begin{bmatrix} -5 \\ 4 \\ 2 \end{bmatrix} \tag{135}$$

The fact that we obtained the exact solution after only 3 iterations should come as no surprise, since Lemma 10.5 guarantees such an outcome. It is worth noting, however, that vectors $x(0)$, $x(1)$ and $x(2)$ differ significantly from the final result, so the process cannot be terminated earlier in this case.

Parallelization of the Conjugate Gradient Method

In order to examine how the conjugate gradient process can be parallelized, let us assume that A is an $n \times n$ matrix, and that we have n processors at our disposal. We will further assume that after the k-th iteration processor i has vectors $x(k)$ and $y(k-1)$ in their entirety, as well as the i-th row of matrix A and component $g_i(k-1)$ of vector $g(k-1)$.

The operations performed by processor i in iteration $k+1$ can then be broken down into five distinct steps, which are described below.

STEP 1. Processor i must first calculate the i-th component of vector $g(k)$. Since $x(k)$ and the i-th row of matrix A are already available, it can compute

$$g_i(k) = [Ax(k)]_i \tag{136}$$

with no additional information. The computation itself entails n multiplications and n additions.

STEP 2. The next step involves computing $\sigma_{k-1}(k)$. Since calculating

$$\sigma_{k-1}(k) = \frac{g^T(k)g(k)}{g^T(k-1)g(k-1)} \tag{137}$$

requires two scalar products, we know that this can be done in time proportional to $\log n$ if we utilize all the available processors. Processor i contributes to this task by evaluating $g_i^2(k)$ and $g_i^2(k-1)$. Once a designated processor gathers all the necessary information, it can compute $\sigma_{k-1}(k)$ and can distribute this value to all other processors (which entails a single node broadcast).

STEP 3. In step 3, processor i updates its component of $y(k)$ as

$$y_i(k) = -g_i(k) + \sigma_{k-1}(k)y_i(k-1) \tag{138}$$

and sends it to all other processors. Note that this requires a multinode broadcast, whose execution time is proportional to $n/\log n$ on a hypercube.

STEP 4. Computing $\rho(k)$ as

$$\rho(k) = -\frac{y^T(k)g(k)}{y(k)^T Ay(k)} \tag{139}$$

entails two scalar products. If we use n processors for this purpose, processor i will contribute by computing terms $y_i(k) \cdot g_i(k)$ and $y_i(k) \cdot [Ay(k)]_i$. This is not a difficult to do, since vector $y(k)$ is available to all processors after Step 3.

STEP 5. In the final step, processor i computes its own component of $x(k+1)$ as

$$x_i(k+1) = x_i(k) + \rho(k)y_i(k) \tag{140}$$

Once this is done, a multinode broadcast is needed to ensure that each processor has vector $x(k+1)$ before the next iteration begins.

From this analysis we can conclude that the most time consuming algebraic operations for processor i are the scalar products needed for computing $g_i(k)$ (in Step 1) and the i-th component of $Ay(k)$ (in Step 4). Since these tasks are executed by a *single* processor,

the computation time is proportional to n. The time needed to calculate $g^T(k)g(k)$, $g^T(k-1)g(k-1)$, $y^T(k)Ax(k)$ and $y(k)^TAy(k)$ is considerably shorter, because these scalar products are determined using n processors.

The communication tasks are dominated by the two multinode broadcasts (which occur in Steps 3 and 5). Given that each of these tasks requires time proportional to $n/\log n$, we can say that a single iteration of the conjugate gradient method can be executed on a hypercube with n processors in time proportional to n. Recalling that the iterative sequence is guaranteed to converge after no more than n steps, the total execution time can now be bounded as $t_{ex} \leq \mu n^2$ (where μ is a constant). This is a clearly an improvement over serial LU factorization, where the number of operations is proportional to n^3 for dense matrices.