Iterative Solvers for Linear Equations, from the Past to the Future

Rüdiger Weiss, Isabella Podgajezkaya, Hartmut Häfner, Willi Schönauer
Numerikforschung für Supercomputer, Rechenzentrum
Postfach 6980, 76128 Karlsruhe, Germany
e-mail: weiss@rz.uni-karlsruhe.de

Abstract

The numerical simulation of many technical and scientific problems leads to the solution of extremely large and sparse linear systems. Advanced computer architectures – vector and parallel computers – and state-of-the-art algorithms have to be used in order to solve these systems with a sufficient accuracy in a reasonable time. Importantly, the simulation of many problems is only possible by the combination of technical and algorithmic progress.

Classical solvers for symmetric and positive definite matrices will be reviewed. From this starting point it will be shown that modern solvers rely on the same principles. With this knowledge the methods can be easily classified despite of their confusing variety. Moreover, it will be shown how to parallelize modern solvers. Thus, the efficient use of advanced computer architectures is combined with modern algorithms to achieve a high performance.

1 Introduction

Many technical and scientific problems can be described by systems of partial differential equations. These equations are usually linearized and discretized by finite differences, finite elements or boundary elements. For all these methods a linear system has to be solved as an inner kernel. The solution of linear systems is also required for applications that do not arise from partial differential equations, for example for the simulation of electric networks or for the design of computer chips.

Direct iterative solvers applied to sparse matrices, resulting from the finite difference and finite element method, produce fill-in, so that even the memory capabilities of supercomputers are insufficient. Therefore, iterative solvers have to be applied. For full matrices, resulting for example from the boundary element method, iterative techniques may be much faster than direct methods if a few digits of accuracy are sufficient – as typical for engineering applications. The dimension of the systems may be extremely large for fine discretizations.

For the numerical solution of these equations with a sufficient accuracy, both extremely fast computers and the most up-to-date methods from numerical mathematics are necessary, i. e. it is only their combination which makes the computation of complex processes possible.

The performance of computers has increased from 1960 until today by a factor of ca. 10^6. This was achieved in the last 10 years by the use of advanced computer architectures like vector and parallel computers. In the mathematical methods we have an increase of ca. 10^4 from 1800 until today. Here conjugate gradient and multi-grid methods have played an important part during the last years. In the combination of computers and algorithms we thus have a performance improvement by a factor of 10^{10}, and only this makes the simulation of many technical and scientific problems possible. The use of the fastest computers without efficient methods as well as the application of the most modern algorithms without modern computers would not lead to results within an acceptable time. Just imagine that instead of an hour, a computation on the fastest computers would take 10^4 hours, i. e. more than a year, without modern algorithms. For such a long time nobody could even guarantee the uninterrupted operation of a computer.

2 Classical Methods

Our problem is to solve the linear system

$$Ax = b. \tag{1}$$

The matrix $A \in \mathbb{R}^{n \times n}$ is a real, square matrix of dimension n and $x,\ b \in \mathbb{R}^n$. Though many of the presented methods work also for singular matrices we assume that A is not singular in order to get a unique solution and unique error estimates.

Starting from an initial guess x_0 an iterative solution method constructs a sequence of approximations $x_0 \to x_1 \to x_2 \to x_3 \to x_4 \to \ldots$ so that $x_k \to x$ for $k \to \infty$. The recurrence can be very sophisticated and non-linear. The residuals are defined by

$$r_k = Ax_k - b\ , \tag{2}$$

and the errors by

$$e_k = x_k - x\ . \tag{3}$$

Methods for symmetric and positive definite systems are discussed in this section. The method of steepest descent is fundamental for many modern iterative techniques. The idea of the method of steepest descent, proposed by Temple [20], is to minimize the quadratic form

$$F(z) = \frac{1}{2}(Az - b)^T A^{-1}(Az - b)\ . \tag{4}$$

If A is symmetric and positive definite, then A^{-1} is symmetric and positive definite. Therefore, $F(z) \geq 0$ for all z and $F(z) = 0$ if and only if $Az - b = 0$. Therefore, the minimum of (4) is the solution of (1).

The gradient of (4) is precisely $Az - b$. The gradient is orthogonal to the isolines of (4) and, therefore, in the direction of the steepest descent. The gradient at the iteration step $k - 1$ is $Ax_{k-1} - b = r_{k-1}$. It is quite natural to choose

$$x_k = x_{k-1} + \delta_k r_{k-1} \tag{5}$$

and to determine δ_k by a one-dimensional minimization so that

$$F(x_k) = \min_{\delta_k} F(x_{k-1} + \delta_k r_{k-1}). \tag{6}$$

Equation (6) is equivalent to $\delta_k = -\frac{r_{k-1}^T r_{k-1}}{r_{k-1}^T A r_{k-1}}$. The update vector $\delta_k r_{k-1} = \delta_k(Ax_{k-1} - b)$ is a multiple of the gradient of the functional (4) in the point x_{k-1} and, therefore, orthogonal to the isoline $F(z) = F(x_{k-1})$. Thus the method proceeds in the direction of the steepest descent. Moreover, the update direction is parallel to the isoline $F(z) = F(x_k)$ because of the minimization property (6); see in figure 1 how the method proceeds from the old to the new iterates. The method converges, if A is positive definite.

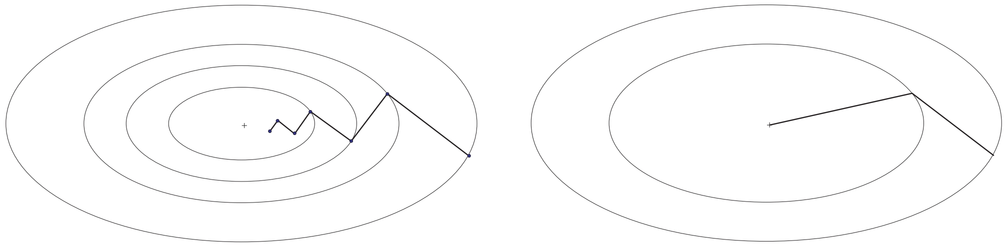

Figure 1: Proceeding of the steepest descent (left) and the classical CG method (right)

Note that the square root of

$$2F(x_k) = r_k^T A^{-1} r_k = e_k^T A e_k = \|e_k\|_A^2 \tag{7}$$

is called energy norm of the error because for systems arising from structural analysis this quantity is equivalent to the potential energy of the discretized problem. It would be meaningful to minimize the Euclidean norm of the error $\|e_k\|$. But for this choice the coefficient δ_k in (5) cannot be determined because the unknown error of the previous step is needed. Another feasible possibility is to minimize the residual norm $\|r_k\|$.

The conjugate gradient (CG) method was developed by Hestenes and Stiefel [9] in the early 1950s for the solution of linear systems with a symmetric, positive definite matrix. The method can be considered as a direct method because the exact solution is obtained at least in the step n in the absence of roundoff errors. Until the early 1970s the method was not widely used. With the increasing use of computers and the consideration as an iterative method by Reid [11] the method recovered importance.

The idea of the classical conjugate gradient method is to minimize the quadratic form (4) by using a multi-dimensional minimization improving the steepest descent method. For the CG method choose

$$x_k = x_{k-1} + \sum_{i=1}^{k} \delta_{i,k} r_{i-1}. \tag{8}$$

The $\delta_{i,k}$ are determined from

$$F(x_k) \quad = \quad \min_{\delta_{1,k},\ldots,\delta_{k,k}} F(x_{k-1} + \sum_{i=1}^{k} \delta_{i,k} r_{i-1}) \tag{9}$$

If the ellipses of the isolines of $F(z) = \textit{constant}$ have very different semi-axes the convergence of the steepest descent method may be poor, while the more sophisticated update direction in (8) improves the convergence. In figure 1 the proceeding of classical CG is depicted in comparison with the steepest descent method. For this two-dimensional example the exact solution is achieved in the second iteration step for classical CG because of the two-dimensional minimization (9).

We get in the absence of rounding errors the exact solution in the iteration step n. However, a considerable reduction of the residuals and errors is generally obtained in far less iteration steps.

We obtain from (9) the orthogonality of the residuals

$$r_k^T r_{j-1} = 0 \tag{10}$$

for $j = 1, \ldots, k$. Thus, the minimization condition (9) is equivalent to the orthogonality conditions (10). Instead of using the minimization property for the definition of CG the orthogonalization conditions can be used. The orthogonalities can be considered as weak formulation for the condition that the residual r_k vanishes for the true solution.

Equation (8) suggests that the length of the recurrence increases with the iteration but the new iterate of classical CG can be calculated by a short recurrence, i. e. two 2-term recurrences.

3 Modern Methods

The method of steepest descent and the classical CG method are the basis for modern methods. These two methods are suited for matrices that are diagonally dominant or symmetric and positive definite. Of course the methods can also be applied to other systems, but the convergence is not guaranteed or may be very slow. Starting from the 1970s the classical CG method has been generalized in order to obtain convergent techniques for non-symmetric and non-positive definite systems. All these methods can be described by the definition of orthogonalization methods [24].

An *orthogonalization method* is defined to be an iterative method satisfying the following relations. For $k \geq 1$

$$x_k \in \tilde{x}_k + \operatorname{span}(q_{k-\sigma_k,k}, \ldots, q_{k-1,k}), \tag{11}$$

where $\sigma_k \leq k$, $\tilde{x}_k \in \operatorname{span}(x_{k-\sigma_k}, \ldots, x_{k-1})$. The vectors $q_{k-i,k} \in I\!\!R^n$ are called *search directions.* The orthogonality condition

$$r_k^T Z_k q_{k-i,k} = 0 \tag{12}$$

is satisfied for $i = 1, \ldots, \sigma_k$. Z_k are auxiliary, non-singular *orthogonalization matrices.*

Different methods result from different choices of the search directions $q_{k-i,k}$, of the orthogonalization matrices Z_k, of the number of search directions σ_k and of $\tilde{x}_k$. In general $\tilde{x}_k = x_{k-\sigma_k}$ or $\tilde{x}_k = x_{k-1}$ is valid.

Equation (12) can be considered as weak formulation for the condition that the residual is vanishing for the true solution. Equation (12) is a generalization of (10) for classical CG, where $Z_k = I$ and $q_{k-i,k} = r_{k-i}$. For classical CG this orthogonalization condition is equivalent to a minimization property. We will see later that under certain conditions a minimization property follows from (12) as well. However, such a minimization condition does not follow always. On the other hand from a minimization property orthogonalization properties can be always derived. Thus the orthogonality condition is more general.

Despite the generality of the orthogonalization method definition some distinct convergence properties can be derived. The next theorem (see [24]) is a fundamental convergence estimate that can be used for all methods. It can be further specialized for special methods involving the particular search directions and norms.

Theorem 3.1 *If $Z_k A^{-1}$ is positive real, i. e. the symmetric part of $Z_k A^{-1}$ is positive definite, then*

$$\|r_k\|_{Z_k A^{-1}} \leq \sqrt{1 + \frac{\rho^2(R)}{\mu_m^2(M)}} \min_{\eta_1,\ldots,\eta_{\sigma_k}} \left\| \sum_{i=1}^{\sigma_k} \eta_i A q_{k-i,k} + \tilde{r}_k \right\|_{Z_k A^{-1}} \tag{13}$$

$$\|e_k\|_{A^T Z_k} \leq \sqrt{1 + \frac{\rho^2(R)}{\mu_m^2(M)}} \min_{\eta_1,\ldots,\eta_{\sigma_k}} \left\| \sum_{i=1}^{\sigma_k} \eta_i q_{k-i,k} + \tilde{e}_k \right\|_{A^T Z_k} \tag{14}$$

holds for orthogonalization methods, where $\tilde{r}_k = A\tilde{x}_k - b$ and $\tilde{e}_k = \tilde{x}_k - x$. $\rho(R)$ is the spectral radius of the skew-symmetric part R of $Z_k A^{-1}$. $\mu_m(M)$ is the minimum eigenvalue of M, the symmetric *part of Z_kA^{-1}. In particular if Z_kA^{-1} is symmetric, then*

$$\|r_k\|_{Z_kA^{-1}} = \min_{\eta_1,\ldots,\eta_{\sigma_k}} \left\| \sum_{i=1}^{\sigma_k} \eta_i A q_{k-i,k} + \tilde{r}_k \right\|_{Z_kA^{-1}}, \qquad (15)$$

$$\|e_k\|_{A^TZ_k} = \min_{\eta_1,\ldots,\eta_{\sigma_k}} \left\| \sum_{i=1}^{\sigma_k} \eta_i q_{k-i,k} + \tilde{e}_k \right\|_{A^TZ_k}. \qquad (16)$$

Theorem 3.1 states a quantitative and a qualitative convergence estimate. The speed of convergence is mainly determined by the size of the norm expression in (14) if the factor of the square root is not too large. This means that the search directions $q_{k-i,k}$ determine the quantitative speed of convergence. The quality of the convergence is given by the norm induced by Z_k. The norm may force a smooth or an oscillating error or residual reduction; see the investigation of special Krylov subspace methods. Moreover, the norm can cause a monotone decrease of the errors or of the residuals as discussed hereafter. A reasonable norm may not force a faster convergence if the true solution is not in the shifted space spanned by the search directions. On the other hand, a badly chosen norm may deteriorate a careful choice of search directions. Thus the quantity of convergence given by the search directions and the quality given by the matrices Z_k are equally important.

A very important result of Theorem 3.1 is that the methods minimize certain quantities if Z_kA^{-1} is positive real. If the search directions are in the Krylov space, $Z_k = I$ and A is symmetric and positive definite, then classical CG is recovered and the energy norm is minimized; see (7). If $Z_k = A$, then the residuals are minimized in the Euclidean norm.

The motivation for introducing the concept of orthogonalization methods was not to obtain a classification of existing methods, but to get new ideas for new, efficient and robust methods. One of these methods is GMERR [23]. The choice $Z_k = A^{-T}$ leads to minimum error methods; see (16). These methods can be implemented without knowing A^{-T} by a trick and are described in the next section.

Other new approaches for linear solvers resulting from the general concept are given in [22]. Moreover, the orthogonalization method concept has been generalized to solution methods for nonlinear equations [25]. In the definition of these methods the residual r_k in (12) has to be substituted by a more general function, but the rest of the definition remains the same. Theoretical results for this generalization are still ahead, but the numerical tests are promising.

4 Krylow-Unterraummethoden

Die am weitesten verbreiteten Methoden zur Lösung von linearen Gleichungssystemen sind Krylow-Unterraummethoden. Zu ihnen gehören auch die klassischen Methoden des steilsten Abstiegs und die konjugierten Gradienten. Zur Zeit haben sich Varianten von van der Vorsts BiCGSTAB [2, 7, 18, 19, 21] als besonders effizient erwiesen. Leider sind gerade für diese Methoden die Orthogonalisierungsmatrizen Z_k noch unbekannt. Sie basieren auf den biconjugierten Gradienten [10] von Lanczos. Die geglätteten Versionen Schönauers BICO [17] und Freund und Nachtigals QMR [4] sind Varianten, die sich durch monoton oder quasi-monoton fallende Residuen auszeichnen. Die Orthogonalisierungsmatrizen Z_k bestehen aus den Schattenvektoren des Lanczos Prozesses.

Eine populäre Methode ist GMRES [13] bei der $Z_k = A$ gilt. Es wird also nach (15) das Residuum in der Euklidschen Norm minimiert. Die Suchrichtungen sind aus dem Krylowraum K_k:

$$q_{k-i,k} \in K_k(A, r_0) = \mathrm{span}(r_0, Ar_0, \ldots A^{k-1}r_0). \qquad (17)$$

Die Fehler e_k und Residuen r_k iterativer Verfahren sind über

$$r_k = Ae_k \tag{18}$$

verknüpft. Für die Euklidschen Normen der Fehler und Residuen gilt

$$\|r_k\| \leq \|A\| \; \|e_k\|, \tag{19}$$
$$\|e_k\| \leq \|A^{-1}\| \; \|r_k\|. \tag{20}$$

Aus (19) und (20) folgt für die relativen Fehler und Residuen

$$\frac{1}{\kappa}\frac{\|r_k\|}{\|r_0\|} \leq \frac{\|e_k\|}{\|e_0\|} \leq \kappa\frac{\|r_k\|}{\|r_0\|} \leq \kappa^2\frac{\|e_k\|}{\|e_0\|}, \tag{21}$$

wobei $\kappa = \|A\| \; \|A^{-1}\|$ die Konditionszahl von A ist. Wenn κ in der Größenordnung von 1 ist, dann sind Fehler und Residuen miteinander gekoppelt. D. h. kleine Residuen bedeuten auch kleine Fehler und umgekehrt. Bei großem κ ist es möglich, daß die relativen Residuen klein sind, während die relativen Fehler groß sind. Eine solche Entkopplung fällt aber nicht auf, da die Fehler nicht berechnet werden können. Verfahren, die die Residuen minimieren sind dann gefährlich, weil der Benutzer von guten Approximationen ausgeht.

GMERR [23] verwendet $Z_k = A^T$, so daß nach (16) die Fehler minimiert werden. Durch die geschickte Wahl der Suchrichtungen in einem geeigneten Krylow-Unterraum

$$q_{k-i,k} \in K_k(A^T, A^T r_0) = \text{span}\left(A^T r_0, \left(A^T\right)^2 r_0, \ldots \left(A^T\right)^k r_0\right) \tag{22}$$

wird erreicht, daß $Z_k = A^{-T}$ niemals berechnet werden muß. Die benötigten Koeffizienten lassen sich ohne Invertierung der Matrix A^T berechnen. Für symmetrische Matrizen reduziert sich das Verfahren auf das Verfahren von Fridman [5]. Eine stabile Implementierung von GMERR wird in [12] angegeben. Unglücklicherweise zeigen Ehrig und Deuflhard in [3], daß das Verfahren für jede rechte Seite nur genau dann konvergiert, wenn die Matrix A normal ist. Für restartete und abgeschnittene Varianten von GMERR gibt es bislang noch keine Theorie, aber vielversprechende Ergebnisse [12].

5 Parallelization

The fastest mathematical algorithm may waste so much CPU-time of the fastest and most advanced available computer that it performs worse than on an ordinary "slow" machine – if it is not implemented properly for the special computer architecture. Most of the here described methods use as basic operations matrix-vector multiplications (mvm), reduce operations like dot products, triadic vector operations of the form *vector = vector + scalar ∗ vector*. For an efficient implementation of the methods on supercomputers it is sufficient to care about these three operations.

The efficient implementation of these operations is standard for vector computers [14]. Therefore, we will focus on parallel systems with distributed memory. For parallel computers there are rules that should be followed to get efficient codes based on the hardware design [16, 15]. We will describe a concept for the message passing programming style. This technique was chosen because it is available for all massively parallel computers and because the distribution of data and operations can be controlled quite efficiently. It may be quite easier to use High Performance Fortran or to transfer dusty decks into it, but usually the only way to achieve acceptable efficiency is to use message passing.

The main principle for communication between two independent processes is to hide the communication behind the calculation. This technique is called latency hiding. It only can be achieved if asynchronous communication is supported by the computer manufacturer. The amount of communication that can be hidden depends first on the volume-to-surface ratio of the algorithm and second on the balancing factor of the computer system. The volume-to-surface ratio is the ratio between the computation units and the communication units, which can be overlapped, and depends on the chosen algorithm. The balancing factor is the ratio between the speed of the communication network and the arithmetic performance on a single processor and depends on the manufacturer. Thus a software engineer can enhance the amount of communication that can be hidden by the choice of an algorithm with a better volume-to-surface ratio - if possible - and by the optimal exploitation of the inherent asynchronity in the chosen algorithm.

For parallel computers it is a desirable property that a program should consume the same CPU-time per processor if the computational amount is increased in the same way as the number of processors. This property is called scalability regarding to the computation time and it guarantees that arbitrarily large problems could be processed in reasonable time if enough processors were available.

It should be mentioned that most of the iterative methods considered as complete algorithm are not scalable regarding to the computation time because usually the number of iterations needed to achieve a certain accuracy increases with the dimension of the system. But the goal is to achieve scalability of the basic operations of iterative methods: matrix-vector multiplication, the reduce operations and the triadic operations.

Note that in this definition the number of unknowns (operations) increases with the number of processors following an approach of Gustafson [6]. We think that this definition is meaningful from a scientific and a practical viewpoint. There is no reason to increase the number of processors if the problem can be solved with a smaller number of processors. The reason to increase the number of processors is that the problem size becomes so large that the execution time exceeds some bound.

There is also a different definition for scalability regarding to the computation time where the problem size is constant, but the number of processors increases. This definition becomes meaningful if a series of problems should be solved in less time, for example to get a reasonable turn-around. However, in this case the ratio between communication with start-up times and computation increases and, thus from a particular number of processors, the speed-up decreases. Moreover, a limit for speed-up is given by Amdahl's law [1].

There is an additional desirable property for parallel computers that a program should consume the same amount of memory per processor if the total memory requirement is increased in the same way as the number of processors. This property is called scalability regarding to the memory and it guarantees that problems which are memory bound could be processed by enlarging the number of processors.

The program package LINSOL [8] achieves scalability regarding the memory by a block-wise distribution of all vectors and a row-block-wise distribution of the matrix onto the p processors. Figure 2 gives an example of a sparse matrix distributed to 4 processors. The matrix A is split row-wise into matrix stripes A^i of length n_i, $i = 1, \dots, p$ (note that generally $n_j \neq n_i$ for $j \neq i$). The $n_i \times n$ matrix A^i is stored in the memory of processor i.

Additionally to this physical subdivision a logical subdivision of each stripe $A^i, i = 1, \dots, p$ into p column blocks $A^{i,1}, \dots, A^{i,p}$ following the subdivision scheme for the rows is done; see Figure 2.

If the matrix A is sparse, the block matrices $A^{i,j}$ will be sparse matrices in general, too. Some will contain only zero entries, called a zero block. Apart from that holds that the (block) matrix-vector multiplication is independent of the special storage format of the matrix. It can be applied to any storage format, e. g. for diagonally stored finite difference matrices, for row-wise stored matrices with column index, for column-wise stored matrices with row-index, etc. It is sufficient to define for each storage pattern the corresponding block matrix-vector multiplication.

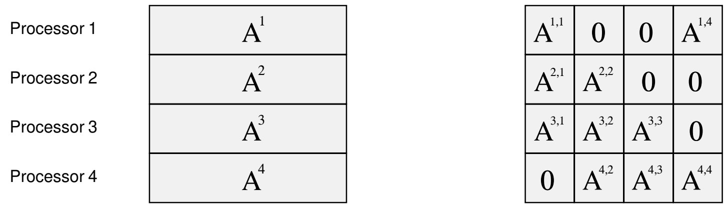

Figure 2: The physical (left) and logical (right) distribution of a matrix to 4 processors

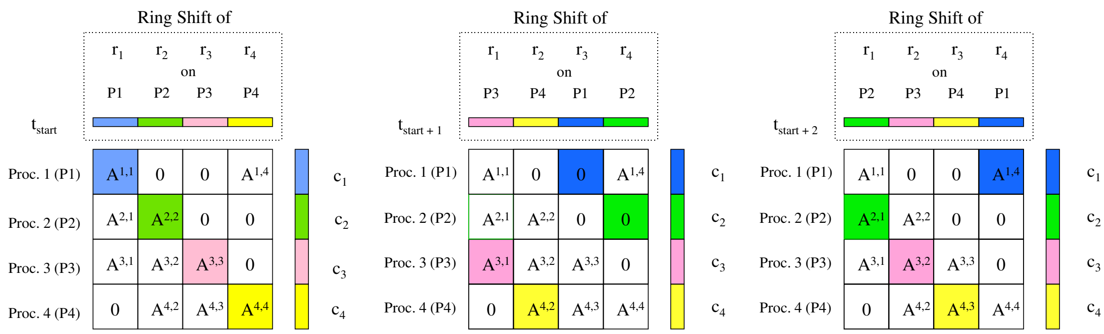

Figure 3: The three computation steps of the matrix-vector multiplication on 4 processors

For the matrix-vector multiplication $c = Ay$ the input vector y and the output vector c are distributed like the rows of the matrix, i.e. processor i has the sub-vectors y^i and c^i of length n_i. The problem is that processor i needs for the processing of the block matrices $A^{i,j}, j \neq i$ the entries of sub-vector y^j, which are not available on processor i without communication.

The matrix-vector multiplication $c = Ay$ is computed on each processor $i, i = 1, \ldots, p$ with a logical subdivision of the matrix A^i and a physical subdivision of the vector y in p blocks:

$$c^i = c^i + \sum_{j=1}^{p} A^{i,j} y^j \tag{23}$$

In the Figure 3 the matrix-vector multiplication is depicted on 4 processors. Diagonal matrix blocks (e.g. $A^{1,2}, A^{2,3}, A^{3,4}, A^{4,1}$) are computed concurrently, if at least one of these blocks contains non-zero matrix elements. If all diagonal matrix blocks contain no non-zero matrix elements, the computation of all these diagonal matrix blocks can be skipped, i.e. the number of communication cycles is reduced by one. In the example of Figure 3 one diagonal matrix block can be skipped. Using this optimization strategy for the communication the matrix-vector multiplication runs more efficient, if the number of communication steps is as small as possible. In finite element applications this can be realized by a bandwidth optimized numbering of the global nodes.

We use two buffers $bufs$ and $bufr$: $bufs$ for the part of the input vector, which is used to perform the block multiplication $A^{i,j}y^j$ for all steps j, and $bufr$, in which the input vector for the next step is received.

Before starting the matrix-vector multiplication processor i copies his input vector y^i into the send buffer $bufs$. Then it triggers the sending of this buffer to the processor, which needs this vector part in the next computation cycle, and triggers the receiving of the vector part of y, which

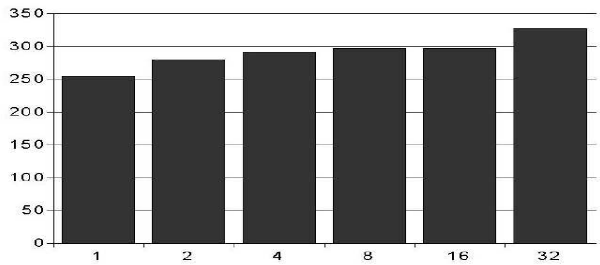

Figure 4: Elapsed time per iteration in ms on the IBM SP

is needed by itself in the next computation cycle. While the communication runs, the multiplication of the current input vector stored in buffer $bufs$ with the sub-matrix $A^{i,j}$ is executed. After the synchronization the pointers to the send and the receive buffers are exchanged and the next cycle starts. In figure 4 the elapsed time per iteration is depicted in dependence of the number of processors or of the problem size, respectively. The linear system results from a 3-dimensional finite element calculation of

$$\nabla k \nabla u = f,$$

where u are displacements and k is the modulus of elasticity. About 2000 unknowns are calculated on each processor. It can be seen that scalability is achieved. However, there is a slight increase of the elapsed time with respect to the number of processors. This increase is caused, among other things, by the dot products that do not scale.

References

[1] G. M. Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485. AFIP Press, Reston, Va., 1967.

[2] C. Brezinski and M. Redivo-Zaglia. Look-ahead in Bi-CGSTAB and other product methods for linear systems. *BIT*, 35:169–201, 1995.

[3] R. Ehrig and P. Deuflhard. GMERR – an error minimizing variant of GMRES. Preprint SC 97-63, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, 14195 Berlin, Germany, 1997.

[4] R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.

[5] V. M. Fridman. The method of minimum iterations with minimum errors for a system of linear algebraic equations with a symmetrical matrix. *USSR Comput. Math. and Math. Phys.*, 2:362–363, 1963.

[6] J. L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[7] M. H. Gutknecht. Variants of BiCGStab for matrices with complex spectrum. *SIAM J. Sci. Comput.*, 14:1020–1033, 1993.

[8] H. Häfner. The program package LINSOL - basic concepts and realization. In A. Sydow, editor, *15th IMACS World Congress on Computation and Applied Mathematics, Berlin, Proceedings, vol. 2, Numerical Methods*, pages 563–568. Verlag Wissenschaft und Technik, 1997.

[9] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–435, 1952.

[10] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bur. Standards*, 49:33–53, 1952.

[11] J. K. Reid. On the method of conjugate gradients for the solution of large sparse systems of linear equations. In J. K. Reid, editor, *Large Sparse Sets of Linear Equations*, pages 231–254. Academic Press, New York, 1971.

[12] M. Rozložník and R. Weiss. On stable implementations of the generalized minimal error method. *J. Comp. and Appl. Math.*, 98(1):49–62, 1998.

[13] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.*, 7(3):856–869, 1986.

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

[15] W. Schönauer. Scientific supercomputing. [Online.] Version 1.0. [Karlsruhe, Germany], 1998. HTML with text as inline images in GIF format. Available from World Wide Web: <http://www.uni-karlsruhe.de/Uni/RZ/Personen/rz03/book/>.

[16] W. Schönauer and H. Häfner. Explaining the gap between theoretical peak performance and real performance for supercomputer architectures. *Scientific Programming*, 3:157–168, 1994.

[17] W. Schönauer, H. Müller, and E. Schnepf. Pseudo-residual type methods for the iterative solution of large linear systems on vector computers. In M. Feilmeier, J. Joubert, and U. Schendel, editors, *Parallel Computing 85*, pages 193–198. Elsevier Science Publishers B. V. (North-Holland), 1986.

[18] G. L. G. Sleijpen and D. R. Fokkema. BiCGStab(l) for linear equations involving unsymmetric matrices with complex spectrum. *Electronic Transactions on Numer. Anal. (ETNA)*, 1:11–32, 1993.

[19] G. L. G. Sleijpen, H. A. van der Vorst, and D. R. Fokkema. BiCGSTAB(l) and other hybrid Bi-CG methods. *Numerical Algorithms*, 7:75–109, 1994.

[20] G. Temple. The general theory of relaxation methods applied to linear systems. *Proc. Roy. Soc. (London)*, A169(A939):476–500, 1939.

[21] H. A. van der Vorst. BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13(2):631–644, 1992.

[22] B. Wagner and R. Weiss. Minimization properties and short recurrences. *Appl. Numer. Math.*, 1999. in print.

[23] R. Weiss. Error-minimizing Krylov subspace methods. *SIAM J. Sci. Comput.*, 15(3):511–527, 1994.

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

[25] R. Weiss and I. Podgajezki. Overview on new solvers for nonlinear systems. *Appl. Numer. Math.*, 1999. in print.