An interactive graphic tool to plot the structure of large sparse matrices

by G. V. Paolini
P. Santangelo

Many engineering and scientific problems involve the solution of large sparse linear systems. To determine an optimal solving strategy for such systems, it is essential to understand the large- and small-scale properties of the associated sparse matrices. We present a graphic tool to analyze the sparsity pattern and the numeric structure of these matrices. Through examples, drawn from our practical experience, we demonstrate the effectiveness and the interactive features of the tool. These features include zooming, scrolling in different directions, sorting of rows and/or columns, and selective plotting, according to the values of the matrix coefficients.

1. Introduction

Solving a large sparse system of linear equations is a common task in many large-scale scientific or engineering computations. Such systems have the form $Ax = b$, where A is a $M \times M$ matrix, b is the known right-hand-side vector, and x is the unknown vector. In problems arising from the discretization of a set of partial differential equations, a given node is only connected to a few other neighboring nodes; consequently, each equation of the system has only a few nonzero elements. Thus, a large number of the coefficients of A are zero, and the matrix is therefore called *sparse*.

With the advent of large supercomputers, it is now possible to solve very large sparse linear problems (i.e., with $M \sim 10^4$–10^6). A number of both direct [1] and iterative [2] algorithms to solve such systems have been developed over time. Their performance is strongly dependent on the specific problem, as well as on the computing facilities currently available. It is therefore necessary to obtain some detailed information about the peculiar properties of a given matrix before choosing the appropriate solving technique.

The experience of the authors has shown that one can gain insight regarding the choice of a solving strategy if one can visualize the sparsity pattern of the matrix. Moreover, even an approximate idea of the values of its coefficients is often useful. It is clear that only a graphic tool can provide all this information quickly.

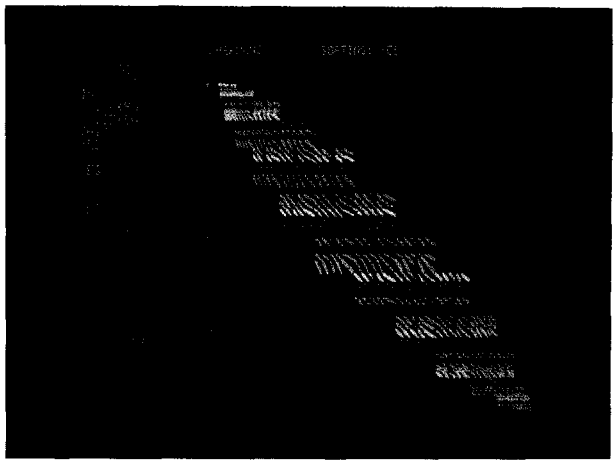

Figure 1

Typical screen displaying the application on a 3179 terminal. The menu on the left contains the white input fields that can be utilized to interactively change the graphic plot.

As a typical example of the utilization of such a tool, the graphic appearance of a block structure for a large matrix immediately suggests that a general solver may not be the most efficient one. In fact, a block structure suggests a "block algebra" as a good solving strategy [3].

An important detail of such a graphic tool is the possibility of analyzing the sparse matrix structure when the matrix is subjected to a given permutation. In fact, some features may appear only for a particular matrix permutation, and the solving strategy may be correspondingly different. For example, matrix permutations are useful in the case of skyline solvers [4] to reduce the bandwidth of the matrix. Another example shows how a particular class of those permutations allows the vectorized solution of sparse triangular systems [5].

In Section 2 we describe the properties of the graphic tool, while in Section 3 we show through examples how these properties can be used to collect relevant information about the matrix.

2. Graphic tool

To analyze the sparsity structure of matrices, a graphic tool has been developed, based on GDDM (Graphical Data Display Manager [6, 7]), which can run under VM or MVS/TSO operating systems and operates on all the graphic devices supported by GDDM (including the 5080 special device).

The tool comprises a subroutine (MATPLO) that can be used through a simple FORTRAN call in a GDDM environment as follows:

```
      .......
C     To initialize GDDM
      CALL FSINIT
      .......
      .......
C     Call to the sparse matrix plotting routine
      CALL MATPLO (list of arguments)
      .......
      .......
C     To terminate GDDM
      CALL FSTERM
      .......
```

where the input arguments, described in detail in [8], refer to the matrix in the standard row-wise format.

When the subroutine is called, a menu is presented on the left side of the screen, indicating the parameters that the subroutine actually uses to plot the matrix. Once these parameters have been selected, the matrix picture is displayed on the screen, as shown in **Figure 1**. (On a 5080 high-resolution device, which uses a dual screen, the menu is shown separately on the alphanumeric screen.) The structure of the matrix is visualized inside a square frame, using a colored symbol for each nonzero element. By default, some given colors are used to tag the elements of the diagonal, and of the lower and upper triangles of the matrix. Alternatively, if sets of elements are to be logically grouped together according to a given criterion, the user can provide additional input information to label with the same color all the matrix elements belonging to a given set.

Many features of the plot can be changed interactively. It is possible to zoom in to the plot by defining a window on the matrix either dynamically with the aid of the cursor or by directly typing the numerical bounds of the submatrix in the proper input fields of the screen. Other functions permit scrolling up and down or diagonally and changing the thickness of the represented elements. If the matrix size is very large (number of rows $\sim 10^5$), plotting can be quite time-consuming, depending on the device utilized. To circumvent this problem, we introduced into the menu an option to represent matrix rows only at regular intervals. This way one can get a coarse representation of the whole matrix, which always proves to be very useful as a starting point for a later and closer analysis.

An important feature of this tool is the capability of supplying a permutation vector to sort the matrix elements. This is useful in testing interactively the effect of the permutation on the sparsity pattern of the matrix. Two cases are allowed:

1. A unique permutation P is specified for both rows and columns, so that the transformed matrix is $\hat{A} = PAP^{-1}$.

2. Two permutations W and P are provided for rows and columns, respectively, so that the transformed matrix is $\hat{A} = WAP^{-1}$. This corresponds to the case in which the solution vector x must be sorted with an ordering different from that of the right-hand-side vector b.

It is worthwhile to note that the colors used to represent matrix elements stay the same under any permutation. This way, after a permutation is applied, it is possible to identify the source of a given element, independently of the coloring strategy. For example, if the default coloring is used, it is possible to tell whether the given element was originally placed in the upper or lower triangle, or along the main diagonal.

Numeric information about the matrix can be interactively obtained by selecting an arbitrary range of values for the matrix coefficients, thus excluding all of the other values from the actual plot: This is particularly useful for the analysis of the numeric structure of the matrix. This feature can be efficiently combined with coloring information to tag with different colors the elements belonging to different numerical ranges.

Another degree of freedom is the capacity to enrich the plot by providing some additional graphics. Features such as a title, lines, or grids can be added to the plot by writing corresponding FORTRAN code in a subroutine with the special name $USER, to be linked just before the MATPLO subroutine. The x and y coordinates must lie within the range 1 to M, M being the dimension of the matrix. Any variable required by the subroutine must be passed through COMMON blocks, shared by the main program and subroutine $USER, while parameters corresponding to the current plot are provided to $USER through a specially named COMMON block.

At any time the content of the plot visualized on the screen can be saved in a file to be successively included in a document or printed separately on a laser printer.

3. Some applications of the tool

We present two examples to demonstrate the usefulness of the graphic tool. The first comes from a large-scale reservoir-modeling study which uses a finite difference scheme. The second one, taken from another reservoir-modeling package, comes from the analysis of the vectorization strategy for an iterative linear system solver [5].

Reservoir modeling is an important source of "difficult" matrices, i.e., of linear systems which are very hard to solve. In this area, the partial differential equations governing the multicomponent oil and water flows are often discretized on a multilayer rectangular domain with relatively coarse horizontal resolution, and with local grid refinements to describe in detail the flow around oil wells. The number of layers is generally much smaller than the number of mesh points in each horizontal direction; thus, the flow has a roughly horizontal nature. Moreover, each space node contains many physical variables such as pressure and saturation of water and the different oil components.

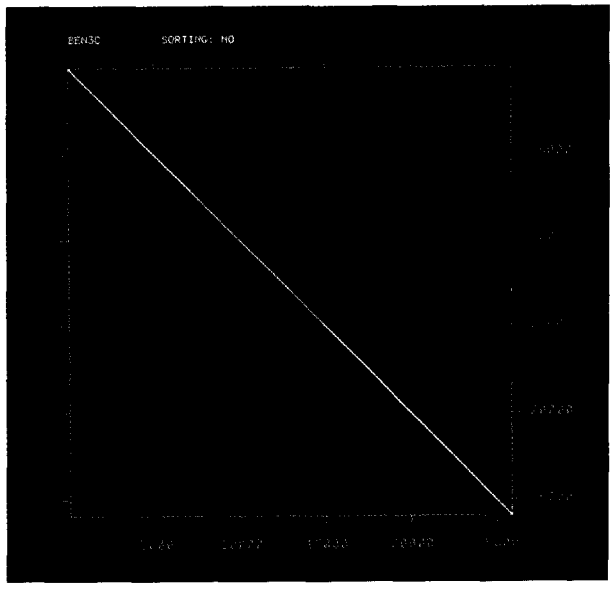

Figure 2

Oil reservoir matrix from a 3D grid. Standard colors are employed: red and blue for the upper and lower triangles, respectively, and white for the main diagonal.

• *Block strategy in reservoir modeling*

Figure 2 represents the matrix of the linear system associated with the structure of a four-layer, double-porosity oil reservoir discretized with finite differences, where the number of nodes in each layer is approximately a thousand. Since the number of simulated partial differential equations is three, there are three variables—the pressure and the saturations of oil and water—at each space point. Thus, we have an estimated amount of about 3 (variables) × 2 (double-porosity) × 4 (layers) × 1000 (nodes) = 24 000 unknowns. The associated system is thus a large one, and represents a challenge for any solving algorithm.

Visual inspection of Figure 2 does not show any interesting pattern, and the enlargement of a portion of the main diagonal, which is shown in **Figure 3**, provides no significant details to help in choosing the appropriate solver. On the other hand, **Figures 4** and **5** show the same matrix after the application of a particular permutation. Here a well-defined block structure, with block length 24, can be detected. This permutation is

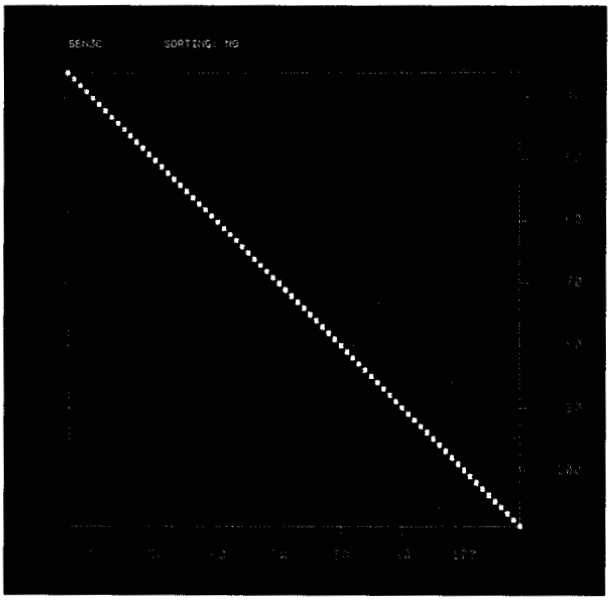

Figure 3

Detail of matrix shown in Figure 2. The enlargement shows small 3 × 3 squares representing the three variables considered: pressure, and saturations of oil and water.

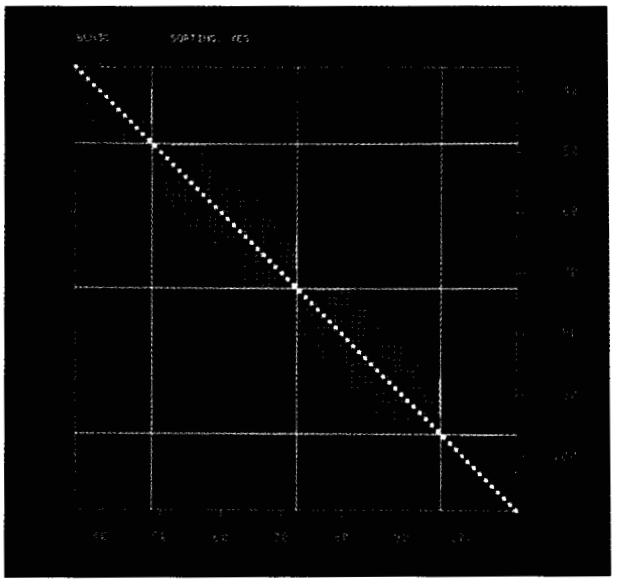

Figure 5

Medium-scale block structure. The enlargement along the diagonal of the sorted matrix in Figure 4 clearly indicates the 24 × 24 block structure given by the permutation. Note the use of subroutine $USER (see Section 2) to delimit the blocks in order to enhance their graphic appearance.

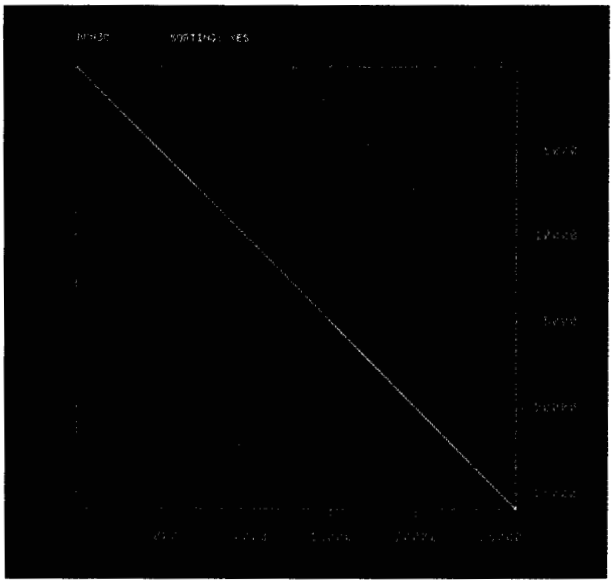

Figure 4

Effect of matrix permutation. The matrix of Figure 2 sorted according to the permutation described in the text. Since standard coloring is used, it is possible to trace the matrix elements back to their initial positions.

obtained by coupling all the variables which are aligned on each vertical column of the three-dimensional mesh. This coupling is a very reasonable one, since in oil reservoirs the flow has a roughly horizontal nature and does not vary strongly along the vertical direction. In particular, the block length may be understood in terms of the above description: 24 = 3 (variables) × 2 (double-porosity) × 4 (layers).

There is another important point. The graphic tool has revealed that a certain physically reasonable permutation organizes the matrix in a block structure; this in turn suggests that the solving strategy must take blocks into consideration. By exploiting blocks, we can take into account the vertical correlation of the variables and therefore control the propagation of errors, thus achieving a faster convergence rate. This is not surprising. Indeed, when the structure of the solution does not vary much for different layers, it is convenient to conceptually group together the nodes across the layers.

We have checked that, in this case, a block successive overrelaxation (BSOR) method gives the shortest total computational time by more than an order of magnitude, when compared to the standard SOR (successive overrelaxation) or even more advanced iterative solvers such as GMRES (generalized minimum residual) or CGS (conjugate gradient squared) [2]. In this case, direct solvers definitely show the worst performance.

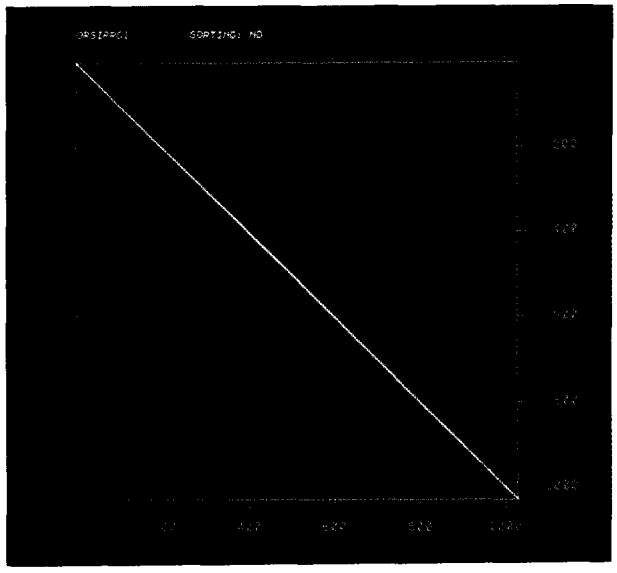

Figure 6

Oil reservoir matrix from a 3D irregular grid. Standard coloring.

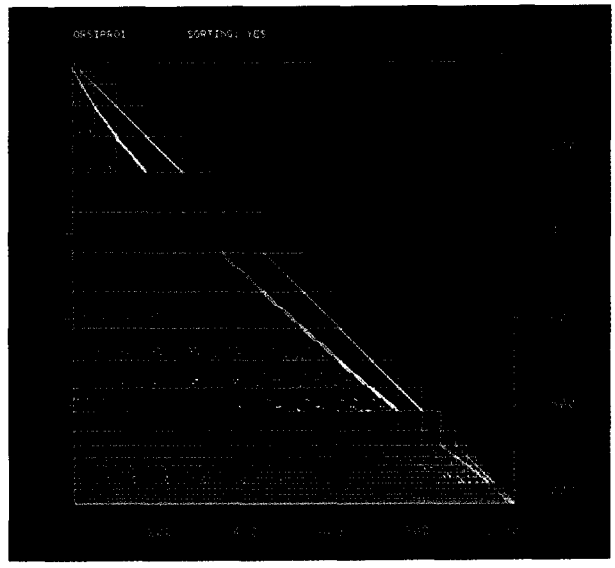

Figure 7

Ordering by wavefronts. The lower incomplete triangular factor of the matrix of Figure 6, after reordering by wavefronts (see Section 3). The grey horizontal lines delimiting the wavefronts were drawn using subroutine $USER. The boxes underneath the main diagonal highlight the underlying block structure. Coloring is also used to tag the different wavefronts.

• *Wavefront technique for sparse linear systems*

Another interesting example of the usefulness of the graphic tool comes from the field of numeric software. A key problem in supercomputing is the vectorization of basic kernels of linear algebra, since these constitute the backbone of many routines, such as solvers for sparse linear systems [9]. Here we consider how the various features of this tool can help in designing and testing an efficient vectorized solver for a linear system.

A common way to solve a linear system is to factor the associated matrix into a lower (L) and an upper (U) triangular matrix such that $A = LU$. This way the problem is transformed into the solution of two triangular systems. In fact, by defining the vector $y = Ux$, one first solves the lower triangular system $Ly = b$, and then the upper triangular one $Ux = y$.

When the system is sparse, one must try to keep the sparsity structure as close as possible to the original system. Among the possible factorizations, a technique known as incomplete LU factorization (ILU) [10] is often used, since this produces factors L and U which have respectively the same sparsity pattern as the lower and upper triangles of the original matrix. While this technique provides only an *approximate* solution of the system, it can be regarded as a good starting point for other solving methods. The ILU technique is used as a *preconditioner* for the linear system $Ax = b$, in order to lower the *condition number* of the problem, i.e., the value $\|A\| \cdot \|A^{-1}\|$, where $\|A\|$ is the norm of matrix A. The modified system $(LU)^{-1}Ax = (LU)^{-1}b$ is then solved, instead of $Ax = b$. If the preconditioner is good, $(LU)^{-1}A$ is better conditioned than A. Preconditioning is often crucial in obtaining a rapid convergence in many iterative algorithms when applied to large linear systems.

Iterative solvers are often used with ILU factorization in a two-step procedure. First, once the proper ILU factorization is computed and verified, an iterative loop begins. At each iteration, preconditioning *inside* the iterative loop requires the solution of an upper and a lower sparse triangular system in addition to the usual operations for the nonpreconditioned sparse solver (such as GMRES or CGS [2]). This method can be efficiently vectorized if the unknowns of such triangular systems are reordered in sets of noninterconnected variables (*wavefronts*) that can be solved concurrently [5].

Once the different wavefronts have been identified [5], the matrix must be rearranged accordingly in order to be able to separate the variables into independent groups, thus allowing for vectorization. Technically this corresponds to sorting the lower (L) and the upper (U) factors with a "lower-wavefront" and an "upper-wavefront" permutation, respectively. **Figures 6** and **7** show the effect of wavefront reordering of the lower triangular part of a matrix coming from an oil reservoir problem discretized on a 3D irregular grid. Figure 6

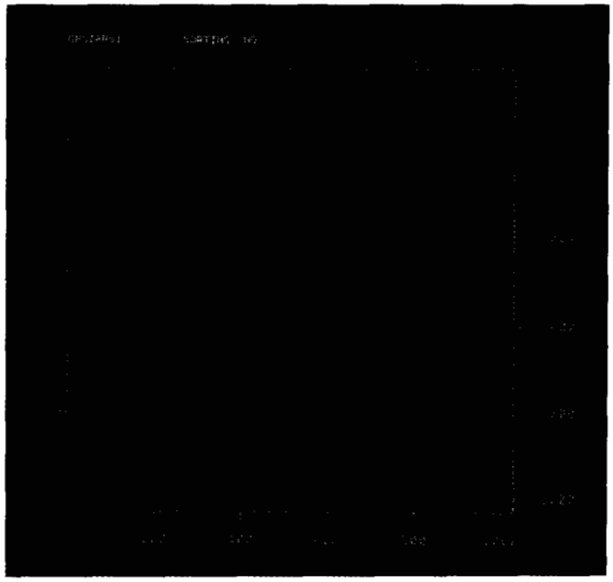

Figure 8

Value selection of matrix coefficients. Elements from the matrix of Figure 6, selected for a specific range ($1–10^3$) of numeric values and displayed by using the interactive features of the visualization tool.

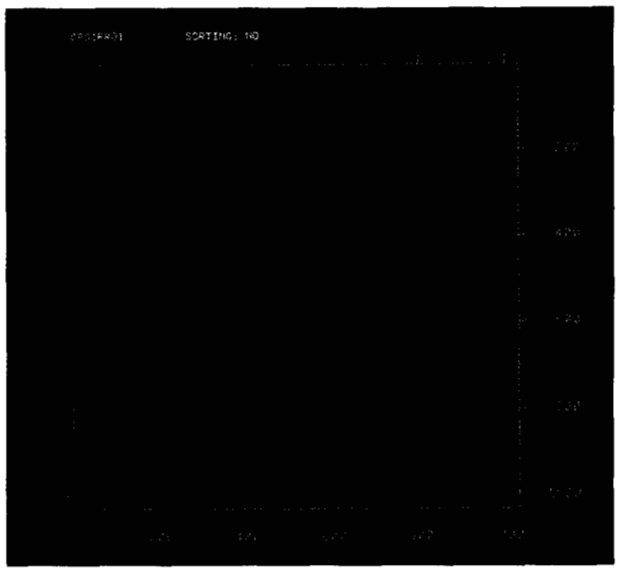

Figure 9

Value selection of matrix coefficients. Display of elements from the matrix of Figure 6 for the range of (10^3–10^6). The remaining elements belong to the diagonal and have negative values.

depicts the matrix with no wavefront reordering (i.e., no permutation). Figure 7 shows its lower triangle after the "lower-wavefront" permutation has been applied. For

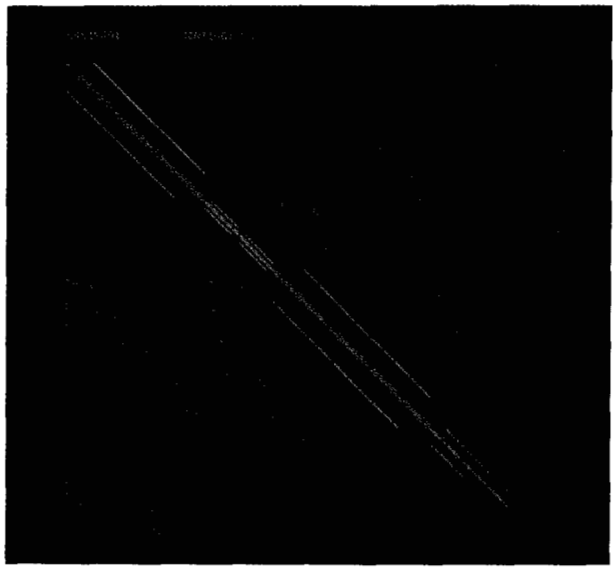

Figure 10

Enhancement of numeric structure by color tagging. Plot of the entire matrix of Figure 6. The elements from Figures 8 and 9 are colored turquoise and yellow, respectively; the diagonal is shown in red.

this plot we have made use of the user-provided routine $USER to obtain the additional horizontal lines that delimit the different wavefronts. Vectorization takes place *inside* each horizontal band, because the variables belonging to the same wavefront do not depend on each other.

Pictorial representation of the effect of wavefront reordering is very important because it gives an immediate idea of the vectorization potential for a given linear system. In other words, the larger the number of elements belonging to the same wavefront (i.e., the dots having the same color in Figure 7), the longer the corresponding vector and the better the achievable performance. Another important point is understanding the extent to which the structure of the matrix is affected by the permutation in terms of bandwidth: This is crucial in minimizing the memory traffic. Figure 7 shows how the algorithm performs well for both of the above points. In fact, graphics immediately shows that the bulk of the reordered matrix has a lower bandwidth than the original one. Moreover, the average length of the wavefronts is large enough to allow a good vectorization of the computational kernel, as we observed in performance tests.

Employing the same oil reservoir matrix, we show in **Figures 8**, **9**, and **10** how it is possible to investigate the numeric structure of a matrix using various techniques

provided by this tool. One method consists in selecting only those values that belong to an arbitrary interval. This can be done interactively using the menu options: Once a given interval is specified changing the input values to the menu, only the chosen range is plotted. We note incidentally that this includes the possibility of selecting the values *external* to a given interval, not just those which are *inside* the specified range. Figures 8 and 9 show the elements of the matrix for two different ranges. The other method exploits the capability of providing color information. Figure 10 contains the same information as the two previous figures, but identifies the different numeric ranges by different colors on the same plot.

Conclusions

We have seen that a key issue in many scientific and engineering problems is the efficient solution of very large sparse linear systems. Once the problem reaches a considerable size, it becomes inconvenient and prohibitive to gain insight into choosing a solving technique solely by inspection of the numeric data. A pictorial representation is then crucial in obtaining synthetic information on the specific problem. To this end an easy-to-use graphic tool based on GDDM has been developed and found to be helpful to analyze interactively the structure of the large sparse matrices encountered in many practical problems. The graphic routine MATPLO, introduced in this paper, provides a useful and flexible tool for investigating the structure of such matrices. Some examples have been presented to demonstrate the idea that a visual inspection of the sparsity and numeric structure of a given matrix may effectively help in choosing and designing the optimal solution strategy for the corresponding linear system.

References

1. I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, London, 1986.
2. L. A. Hagemann and D. M. Young, *Applied Iterative Methods*, Academic Press, Inc., New York, 1981.
3. S. C. Eisenstat, H. C. Elman, and M. H. Schultz, "Block Preconditioned Conjugate Gradient-Like Methods for Numerical Reservoir Simulation," *Proceedings of the SPE 1985 Reservoir Simulation Symposium*, Society of Petroleum Engineers of AIME, Richardson, TX, 1985, Paper No. 13534, pp. 397–405.
4. K. J. Bathe, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.
5. G. V. Paolini and G. Radicati di Brozolo, "Data Structures to Vectorize CG Algorithms for General Sparsity Patterns," *BIT* **29**, 703–718 (1989).
6. *GDDM, Graphical Data Display Manager, Version 2, Release 1, Base Programming Reference, Vols. 1-2*, Order No. SC33-0332, 1986; available through IBM branch offices.
7. *GDDM, Graphical Data Display Manager, Version 2, Release 1, Application Programming Guide, Vols. 1-2*, Order No. SC33-0337, 1986; available through IBM branch offices.
8. G. V. Paolini and P. Santangelo, "A Graphic Tool to Plot the Structure of Large Sparse Matrices," *Technical Report ICE-0034*, IBM European Center for Scientific and Engineering Computing, Rome, Italy, 1989.
9. G. H. Golub and C. F. Van Loan, *Matrix Computations*, North Oxford Academic, Oxford, England, 1983.
10. O. Axelsson, "A Survey of Preconditioned Iterative Methods for Linear Systems of Algebraic Equations, *BIT* **25**, 166–187 (1985).

Received November 6, 1989; accepted for publication September 28, 1990

Gaia Valeria Paolini *IBM European Center for Scientific and Engineering Computing (ECSEC), Via Giorgione 159, 00147 Rome, Italy.* Dr. Paolini received her doctorate in physics from Rome University in 1986. The following year she did postdoctoral work at the IBM Almaden Research Center, on the simulation of solids by means of molecular dynamics techniques. Since June 1988, Dr. Paolini has been a research staff member at the IBM ECSEC in Rome, where she has conducted research on numerical algorithms for the solution of sparse linear systems on vector and parallel machines. She is currently working in the area of dynamic simulation of molecular liquids and biochemical systems.

Paolo Santangelo *IBM European Center for Scientific and Engineering Computing (ECSEC), Via Giorgione 159, 00147 Rome, Italy.* Dr. Santangelo received his doctorate in physics from Rome University in 1979. From 1980 to 1983 he was a researcher at the Astronomical Institute of Rome University, where he worked on the dynamical evolution of groups and clusters of galaxies. In 1984, he joined the IBM Rome Scientific Center, where he worked in the field of mathematical modeling of hydrological systems. Dr. Santangelo is now a permanent research staff member of the IBM European Center for Scientific and Engineering Computing (ECSEC), where he has worked on high-resolution numerical computations for two-dimensional turbulence and three-dimensional flows. In 1986 Dr. Santangelo received an IBM Outstanding Technical Achievement Award for his work on the parallel implementation of the gravitational N-body problem on the ECSEC Loosely Coupled Parallel System. Subsequently he worked on the porting of industrial codes for the simulation of oil reservoirs on the IBM 3090 Vector Facility and in 1989 received an IBM Outstanding Technical Achievement Award for his contribution to that project. Dr. Santangelo has published numerous papers in the field of astrophysics and fluid dynamics. His current interests are in scientific visualization.