INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 6, 913–925 (1986)

FINITE ELEMENT GROUND-WATER MODELS IMPLEMENTED ON VECTOR COMPUTERS

W. PELKA AND A. PETERS

Institute for Hydraulic Engineering and Water Resources Development, Aachen University of Technology (RWTH), D-5100 Aachen, West Germany

SUMMARY

Finite element models, optimized for running on conventional serial computers, are not suitable to make use of the potentional high performance of today's vector or parallel computers. Also, automatic vectorization by the compiler or manual vectorization at the local level do not by far lead to the required and expected computational speed.

A global change of the overall program logic or a complete redesign becomes necessary, i.e. a completely new generation of program systems has to be created, considering the new hardware characteristics and abilities.

A new computer-independent programming technique for finite element problems to be implemented on vector computers is proposed and the test results of scalar and vectorized program structures on a conventional serial and on a non-conventional pipeline computer are discussed.

KEY WORDS Finite Elements Ground-water Models Vector Computers

INTRODUCTION

In the past two decades the finite element method has found increasingly widespread application in nearly all areas of engineering and science. The ability to approximate complex geometries efficiently, the easy incorporation of all types of boundary conditions and the very convenient handling of arbitrary tensorial quantities made it the preferred choice in solving fluid mechanics and transport problems.

In order to implement a numerical algorithm on any computer its adaption to the computer's characteristics is necessary. The actual structure of most finite element programs is determined by the architecture of the third computer generation, based on the von Neumann principle (SISD, single instruction stream and single data stream).

Owing to high complexity, especially when considering time-dependent, non-linear and three-dimensional problems, in many cases an implementation on these computers is not possible or is economically infeasible. Even talking into account future advances in semiconductor design, machines of this architecture will not be able to achieve the computational speed necessary to solve computationally massive problems in engineering and science. Vector computers, based on new hardware architectures, are the great hope to make use of these advanced models in practice. Bench-marks of conventional finite element program systems on SIMD (single instruction and multi data stream) computers, however, proved to be in some way disillusioning.[1]

In the present paper a new computer-independent programming technique for finite element

0271–2091/86/120913–13$06.50

Received 18 January 1986
Revised 9 May 1986

problems to be implemented on vector computers is proposed. The test results of vectorized and scalar program structures on a conventional serial (CDC Cyber 175) and a non-conventional pipeline computer (CDC Cyber 205) are discussed.

SOFTWARE PARTICULARITIES OF VECTOR COMPUTERS

Pipeline or vector computers realize the principle of a single instruction and multi data stream (SIMD). On these computers, when the same operation is to be performed on a series of operands, a vector machine instruction can perform the operation faster than a series of scalar (conventional) machine instructions. The steps of a scalar machine instruction cannot be overlapped and the operation is to be performed only on a single set of values. The conceptual idea behind a vector or pipeline machine instruction is essentially that of an assembly line: if the same arithmetic operation is going to be repeated many times, throughput can be greatly increased by dividing the operation into a sequence of subtasks and maintaining the flow of operand pairs in various states of completion. The more sets of operands are to be processed the more efficient is the vector instruction.

A series of values that are stored in contiguous memory locations and which serves as an operand for a vector machine instruction is called vector. A vector is completely defined by its first element, which must be an array element, its length and data type. The vectorizing compiler recognizes vectorizable loop constructions and generates the respective vector code.[2]

The speed of computation can also be increased, making use of the vector intrinsic functions with vector input/output arguments. The vector intrinsic functions are not standardized, but are implemented on any pipeline computer with similar names.

Important for the vectorization of the finite element structures are the GATHER and SCATTER intrinsic functions. By means of a GATHER function (Figure 1),

$$\vec{B}^{(m)} = \overset{\vec{K}^{(m)}}{\lhd\!\boxed{G}}\ \vec{A}^{(n)}, \tag{1a}$$

all m elements of vector $\vec{B}$ are gathered from the n elements of vector $\vec{A}$, according to the pointer vector $\vec{K}$, containing the respective indices of elements of $\vec{A}$:

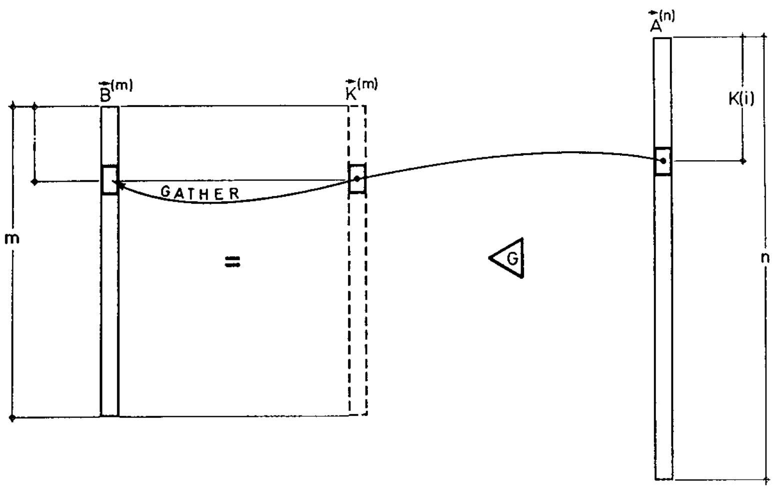


Figure 1. GATHER function (equation (1))

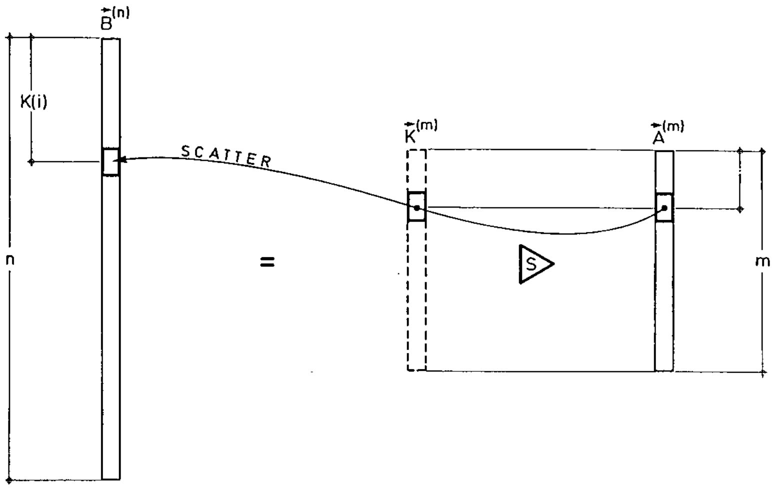


Figure 2. SCATTER function (equation (2))

$$B(i) = A(K(i)), \quad \text{with} \quad K(i) \leqslant n; \quad i = 1, \ldots, m. \tag{1b}$$

Using a SCATTER function (Figure 2),

$$\vec{B}^{(n)} = \boxed{S \triangleright}^{\vec{K}^{(m)}} \vec{A}^{(m)}, \tag{2a}$$

all m elements of vector $\vec{A}$ are scattered to vector $\vec{B}$ of length n, according to the pointer vector $\vec{K}$, containing the respective indices of elements of $\vec{B}$:

$$B(K(i)) = A(i), \quad \text{with} \quad K(i) \leqslant n; \quad i = 1, \ldots, m. \tag{2b}$$

Bit-vectors, whose elements are 1 or 0, are used as logical arguments in vector relational expressions. If the right hand side expression of

$$\overrightarrow{\text{IBIT}}^{(m)} = (\vec{C}^{(m)} \geqslant \vec{B}^{(m)}) \quad i = 1, \ldots, m \tag{3a}$$

is true, IBIT(i) is set to 1, otherwise to 0. The bit-vector may be used later, for example

$$\vec{A}^{(m)} = \begin{cases} \vec{B}^{(m)} & 1 \\ \overrightarrow{\text{IBIT}}(m) & \\ \vec{C}^{(m)} & 0 \end{cases} \tag{3b}$$

to store the vector elements $B(i)$ or $C(i)$ into the element i of $\vec{A}$, depending on the content of IBIT(i). If the element i of $\overrightarrow{\text{IBIT}}$ contains a '1', $B(i)$ is assigned to the coefficient i of $\vec{A}$, otherwise $C(i)$.

A detailed presentation of the FORTRAN software features of vector computers can be found in the specific manuals.[3,4]

TESTED FINITE ELEMENT MODEL

The finite element algorithm of a two-dimensional ground-water model was chosen to test conventional and vectorized programming techniques.

Following the Dupuit–Forchheimer assumptions for large ground-water basins, the two-dimensional horizontal time-dependent ground-water flow is governed by

$$S\frac{\partial h}{\partial t} + \frac{\partial}{\partial x_i}\left(T_{ij}(h)\frac{\partial h}{\partial x_j}\right) + Q(t) = 0, \tag{4}$$

where h is the unknown piezometric head. S and T represent the specific storativity and the transmissivity of the aquifer, and Q is a time-dependent sink or source term.

Boundary conditions are of Dirichlet (prescribed head) or Neumann (prescribed flux) type.

The non-linearity, rising from the dependence of the transmissivity on the actual piezometric head, can be considered by an iteration algorithm, i.e. within the iteration step the linearized equation

$$S\frac{\partial h^{\nu}}{\partial t} + \frac{\partial}{\partial x_i}\left(T_{ij}(h^{\nu-1})\frac{\partial h^{\nu}}{\partial x_j}\right) + Q(t) = 0 \tag{5}$$

will be solved.

The finite element approach, based on a variational principle or weighted residual technique, leads to a linear equation system of the form

$$D_{mn}h_n + E_{mn}\frac{\partial h_n}{\partial t} = F_m, \tag{6}$$

where

$$E_{mn} = \sum_e \int_{A^e} S^e \phi_m^e \phi_n^e \, \mathrm{d}A^e, \tag{6a}$$

$$D_{mn} = \sum_e \int_{A^e} \frac{\partial \phi_m^e}{\partial x_i} T_{ij} \frac{\partial \phi_n^e}{\partial x_j} \mathrm{d}A^e \tag{6b}$$

and the right hand side vector F_m represents the boundary fluxes and sink/source terms.

ϕ are the element basis functions. The integration is performed over the element's area A^e and the equations for all elements e are gathered to form the global equation system.

The numerical model uses for discretization the linear triangular element, which in this case offers a solution of satisfactory accuracy by minimal computing effort.[5,6]

The integration with respect to time is performed as usual by a finite difference scheme.

Solving the system of linear algebraic equations yields the vector of the unknown nodal values of the piezometric head. To obtain the nodal flux balances and the flow velocities, the piezometric head vector is resubstituted into equation (6) and differentiated, respectively.

CONVENTIONAL PROGRAM STRUCTURE

The analysis process of the test model consists mainly of three phases to be carried out for every time step:

(i) calculation of the element matrices and assemblage of the structure matrix
(ii) solution of the equilibrium equations
(iii) calculation of nodal balances and derivatives.

In practice, limits of the available storage capacity and speed of a particular computer constrain one to specific implementations of any of the three phases, with a significant effect on the efficiency of the other phases. Currently, implementations of the first and third phases are based on two approaches.

The element matrices or the derivative matrices are generated and stored on a peripheral storage device for later use, or the derivative matrices and element matrices are regenerated, when required. Common to both approaches is the sequential computation of the element matrices.[7,8]

The reasons for the weak performance of conventional programs on vector computers are obvious. As Figure 3 shows, in a long loop for all elements the element matrices are sequentially computed and stored in the general structure matrix. The inner tasks of the loop require many algebraic operations on vectors of short length, preventing the pipeline processor from developing its full potential.

Owing to the start-up time necessary to begin every vector operation, for short vectors the vectorized computation may take even longer than the equivalent row of scalar operations.[1]

The same problem occurs in the third phase, when computing the nodal balances by resubstitution of the solution vector into the equation system and when evaluating the derivatives such as velocities and fluxes.

The implementation of the second phase consists essentially of solving directly or iteratively a system of linear algebraic equations. Currently solution techniques, used in many conventional

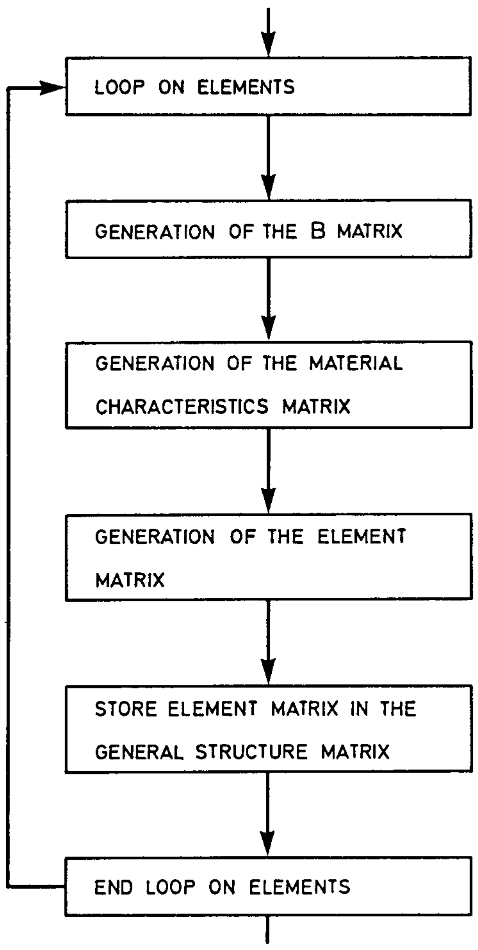

LOOP ON ELEMENTS
GENERATION OF THE B MATRIX
GENERATION OF THE MATERIAL CHARACTERISTICS MATRIX
GENERATION OF THE ELEMENT MATRIX
STORE ELEMENT MATRIX IN THE GENERAL STRUCTURE MATRIX
END LOOP ON ELEMENTS

Figure 3. Sequential computing of the element matrices and assemblage of the general structure matrix

programs, are basically applications of Gaussian elimination.[9] The chosen algorithm has not only to be quick and stable, but to suit to the specific computer features as well.

An effective storage scheme for the symmetric and banded global matrix is to store the coefficients below the skyline of the non-zero elements only. When solving the equations, no coefficients outside the skyline are stored or processed. Considering a vectorized solution of the simultaneous equations, this procedure of addressing coefficients is not well suited, because non-vectorizable index computations are needed.

It can be seen that the finite element programs in their actual form cannot take advantage of the capabilities of the vector computers. A complete redesign becomes necessary.

VECTOR PROGRAM STRUCTURE

The basic idea of effective vector programming for the finite element method is to take advantage of the large data arrays that must be handled in the form of vectors and arrays.

First of all the user-convenient scalar input has to be rearranged in compact vectors that are basic data for the vector machine instructions.

Figure 4 demonstrates the basic procedure. Values such as co-ordinates, evaluated for all n nodes of the finite element mesh are gathered to element node vectors of length m. In this process the system topology vectors ($\vec{K}_1, \vec{K}_2, \vec{K}_3$, in Figure 5), connecting element number and associated node numbers, were used as pointer or filter vectors. The element node vectors of the co-ordinates can easily be used to perform a completely vectorized precalculation of the derivative matrices **B** and the areas of all elements.

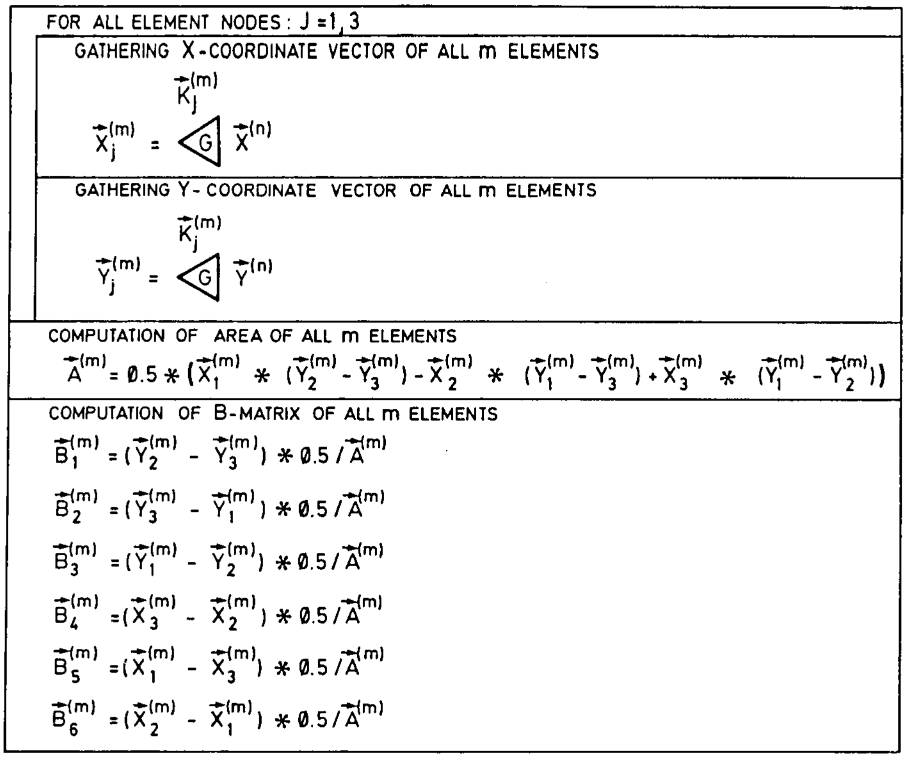

FOR ALL ELEMENT NODES : J = 1, 3
GATHERING X-COORDINATE VECTOR OF ALL m ELEMENTS
$\vec{X}_j^{(m)} = \overset{\vec{K}_j^{(m)}}{\lhd G} \vec{X}^{(n)}$
GATHERING Y-COORDINATE VECTOR OF ALL m ELEMENTS
$\vec{Y}_j^{(m)} = \overset{\vec{K}_j^{(m)}}{\lhd G} \vec{Y}^{(n)}$
COMPUTATION OF AREA OF ALL m ELEMENTS
$\vec{A}^{(m)} = 0.5 * (\vec{X}_1^{(m)} * (\vec{Y}_2^{(m)} - \vec{Y}_3^{(m)}) - \vec{X}_2^{(m)} * (\vec{Y}_1^{(m)} - \vec{Y}_3^{(m)}) + \vec{X}_3^{(m)} * (\vec{Y}_1^{(m)} - \vec{Y}_2^{(m)}))$
COMPUTATION OF B-MATRIX OF ALL m ELEMENTS
$\vec{B}_1^{(m)} = (\vec{Y}_2^{(m)} - \vec{Y}_3^{(m)}) * 0.5 / \vec{A}^{(m)}$
$\vec{B}_2^{(m)} = (\vec{Y}_3^{(m)} - \vec{Y}_1^{(m)}) * 0.5 / \vec{A}^{(m)}$
$\vec{B}_3^{(m)} = (\vec{Y}_1^{(m)} - \vec{Y}_2^{(m)}) * 0.5 / \vec{A}^{(m)}$
$\vec{B}_4^{(m)} = (\vec{X}_3^{(m)} - \vec{X}_2^{(m)}) * 0.5 / \vec{A}^{(m)}$
$\vec{B}_5^{(m)} = (\vec{X}_1^{(m)} - \vec{X}_3^{(m)}) * 0.5 / \vec{A}^{(m)}$
$\vec{B}_6^{(m)} = (\vec{X}_2^{(m)} - \vec{X}_1^{(m)}) * 0.5 / \vec{A}^{(m)}$

Figure 4. Vectorized computation of the B matrices

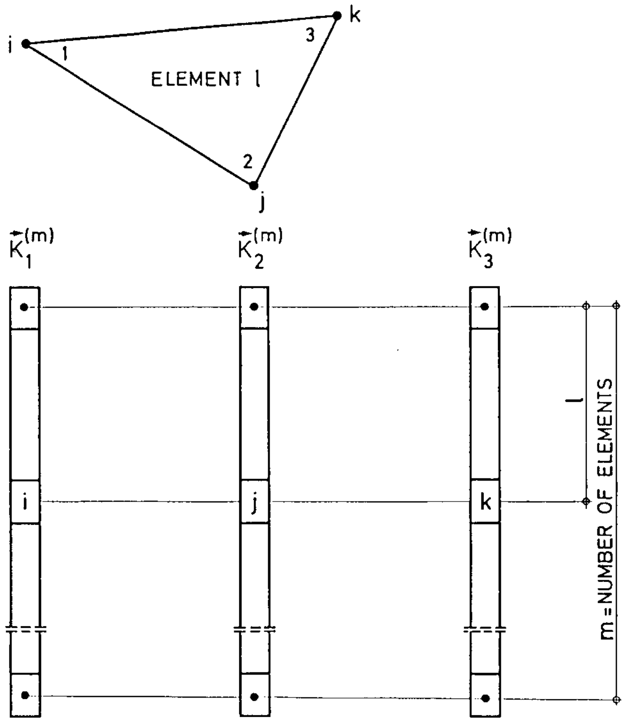


Figure 5. Generation of the topology vectors

A similar procedure is used when converting nodal values to element mean values, as shown in Figure 6 for the bedrock and caprock elevations.

The usual technique of defining certain material types and using a pointer vector, indicating the material type of each element, is very convenient in the sense of minimizing the manual input and the storage requirements, but will prevent later an effective vectorization, when computing the element matrices. By means of a gather instruction, element vectors of permeability and storativity are built, making use of the material type pointer vector (Figure 7). A vectorized preparation of the storativity for later use in the storativity matrix may also take place here.

The classical procedure of stiffness matrix computation and assemblage, sequentially carried out on the elements, has to be split up into subtasks of operations, which can be computed vectorized for all elements simultaneously. Figure 8 shows a completely vectorized computation of the element matrices.

In order to permit the vector processor to develop its full potential, sequential schemes, as shown in Figure 3, have to be converted to a new form. To obtain long vectors, the inner and outer loops are interchanged. By means of this technique the outer non-vectorizable loops are of minor length, whereas all inner loops, and by this all algebraic computations, act on long vectors of length *m*.

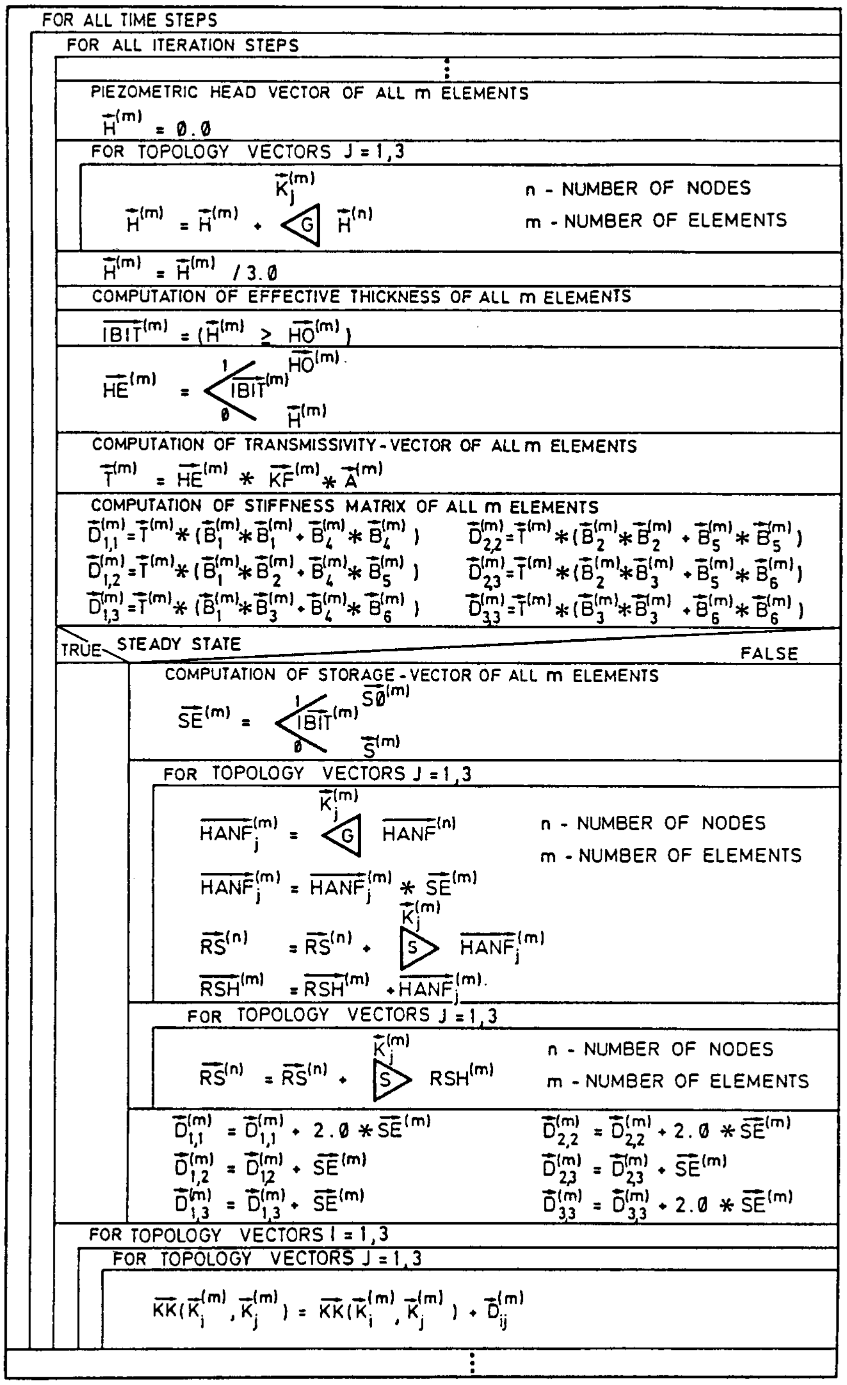
Gathering the actual nodal values of piezometric head to element values and applying the bit-vector technique of logical decisions, a vectorized computation of all elements' transmissivities takes place. The transmissivity vector and the precalculated derivative matrices are used to compute the element matrices by means of vector instructions. The storage matrices are computed in a similar way. Right hand side contributions and element matrices are scattered to the global matrix and right hand side vector, again using the mesh topology vectors as pointer or filter vectors.

```
GATHERING BEDROCK VECTOR OF ALL m ELEMENTS
$\vec{HU}^{(m)} = 0.0$
FOR ALL ELEMENT NODES : J = 1,3
    $\vec{HU}^{(m)} = \vec{HU}^{(m)} + \overset{\vec{K}_j^{(m)}}{\triangleleft G}\ \vec{HU}^{(n)}$
$\vec{HU}^{(m)} = \vec{HU}^{(m)} / 3.0$
GATHERING CAPROCK VECTOR OF ALL m ELEMENTS
$\vec{HO} = 0.0$
FOR ALL ELEMENT NODES : J = 1,3
    $\vec{HO}^{(m)} = \vec{HO}^{(m)} + \overset{\vec{K}_j^{(m)}}{\triangleleft G}\ \vec{HO}^{(n)}$
$\vec{HO}^{(m)} = \vec{HO}^{(m)} / 3.0$
```

Figure 6. Rearrangement of the nodal bedrock and caprock elevations into vectors of length *m*

```
GATHERING PERMEABILITY VECTOR OF ALL m ELEMENTS
$\vec{KF}^{(m)} = \overset{\vec{MTYP}^{(m)}}{\triangleleft G}\ \vec{KF}^{(nmtyp)}$
GATHERING STORAGE VECTORS OF ALL m ELEMENTS
$\vec{S}^{(m)} = \overset{\vec{MTYP}^{(m)}}{\triangleleft G}\ \vec{S}^{(nmtyp)}$   (unconfined)
$\vec{S0}^{(m)} = \overset{\vec{MTYP}^{(m)}}{\triangleleft G}\ \vec{S0}^{(nmtyp)}$   (confined)
$\vec{S}^{(m)} = \vec{S}^{(m)} * \vec{A}^{(m)} / (12.0 * DT)$
$\vec{S0}^{(m)} = \vec{S0}^{(m)} * \vec{A}^{(m)} * (\vec{HO}^{(m)} - \vec{HU}^{(m)}) / (12.0 * DT)$
```

Figure 7. Rearrangement of the material characteristics data input into vectors of length *m*



```
FOR ALL TIME STEPS
  FOR ALL ITERATION STEPS
    :
    PIEZOMETRIC HEAD VECTOR OF ALL m ELEMENTS
      $\vec{H}^{(m)} = 0.0$
    FOR TOPOLOGY VECTORS J = 1,3
      $\vec{H}^{(m)} = \vec{H}^{(m)} + \overset{\vec{K}_j^{(m)}}{\lhd G}\ \vec{H}^{(n)}$      n - NUMBER OF NODES
                                                                   m - NUMBER OF ELEMENTS
    $\vec{H}^{(m)} = \vec{H}^{(m)} / 3.0$
    COMPUTATION OF EFFECTIVE THICKNESS OF ALL m ELEMENTS
      $\overline{IBIT}^{(m)} = (\vec{H}^{(m)} \geq \vec{HO}^{(m)})$
      $\vec{HE}^{(m)} = \overline{IBIT}^{(m)}$ ? 1: $\vec{HO}^{(m)}$ ; 0: $\vec{H}^{(m)}$
    COMPUTATION OF TRANSMISSIVITY-VECTOR OF ALL m ELEMENTS
      $\vec{T}^{(m)} = \vec{HE}^{(m)} * \vec{KF}^{(m)} * \vec{A}^{(m)}$
    COMPUTATION OF STIFFNESS MATRIX OF ALL m ELEMENTS
      $\vec{D}_{1,1}^{(m)} = \vec{T}^{(m)} * (\vec{B}_1^{(m)} * \vec{B}_1^{(m)} + \vec{B}_4^{(m)} * \vec{B}_4^{(m)})$      $\vec{D}_{2,2}^{(m)} = \vec{T}^{(m)} * (\vec{B}_2^{(m)} * \vec{B}_2^{(m)} + \vec{B}_5^{(m)} * \vec{B}_5^{(m)})$
      $\vec{D}_{1,2}^{(m)} = \vec{T}^{(m)} * (\vec{B}_1^{(m)} * \vec{B}_2^{(m)} + \vec{B}_4^{(m)} * \vec{B}_5^{(m)})$      $\vec{D}_{2,3}^{(m)} = \vec{T}^{(m)} * (\vec{B}_2^{(m)} * \vec{B}_3^{(m)} + \vec{B}_5^{(m)} * \vec{B}_6^{(m)})$
      $\vec{D}_{1,3}^{(m)} = \vec{T}^{(m)} * (\vec{B}_1^{(m)} * \vec{B}_3^{(m)} + \vec{B}_4^{(m)} * \vec{B}_6^{(m)})$      $\vec{D}_{3,3}^{(m)} = \vec{T}^{(m)} * (\vec{B}_3^{(m)} * \vec{B}_3^{(m)} + \vec{B}_6^{(m)} * \vec{B}_6^{(m)})$
    STEADY STATE
    TRUE:
    FALSE:
      COMPUTATION OF STORAGE-VECTOR OF ALL m ELEMENTS
        $\vec{SE}^{(m)} = \overline{IBIT}^{(m)}$ ? 1: $\vec{S0}^{(m)}$ ; 0: $\vec{S}^{(m)}$
      FOR TOPOLOGY VECTORS J = 1,3
        $\overline{HANF}_j^{(m)} = \overset{\vec{K}_j^{(m)}}{\lhd G}\ \overline{HANF}^{(n)}$      n - NUMBER OF NODES
                                                               m - NUMBER OF ELEMENTS
        $\overline{HANF}_j^{(m)} = \overline{HANF}_j^{(m)} * \vec{SE}^{(m)}$
        $\vec{RS}^{(n)} = \vec{RS}^{(n)} + \overset{\vec{K}_j^{(m)}}{S \rhd}\ \overline{HANF}_j^{(m)}$
        $\overline{RSH}^{(m)} = \overline{RSH}^{(m)} + \overline{HANF}_j^{(m)}$.
      FOR TOPOLOGY VECTORS J = 1,3
        $\vec{RS}^{(n)} = \vec{RS}^{(n)} + \overset{\vec{K}_j^{(m)}}{S \rhd}\ RSH^{(m)}$      n - NUMBER OF NODES
                                                          m - NUMBER OF ELEMENTS
      $\vec{D}_{1,1}^{(m)} = \vec{D}_{1,1}^{(m)} + 2.0 * \vec{SE}^{(m)}$      $\vec{D}_{2,2}^{(m)} = \vec{D}_{2,2}^{(m)} + 2.0 * \vec{SE}^{(m)}$
      $\vec{D}_{1,2}^{(m)} = \vec{D}_{1,2}^{(m)} + \vec{SE}^{(m)}$            $\vec{D}_{2,3}^{(m)} = \vec{D}_{2,3}^{(m)} + \vec{SE}^{(m)}$
      $\vec{D}_{1,3}^{(m)} = \vec{D}_{1,3}^{(m)} + \vec{SE}^{(m)}$            $\vec{D}_{3,3}^{(m)} = \vec{D}_{3,3}^{(m)} + 2.0 * \vec{SE}^{(m)}$
    FOR TOPOLOGY VECTORS I = 1,3
      FOR TOPOLOGY VECTORS J = 1,3
        $\vec{KK}(\vec{K}_i^{(m)}, \vec{K}_j^{(m)}) = \vec{KK}(\vec{K}_i^{(m)}, \vec{K}_j^{(m)}) + \vec{D}_{ij}^{(m)}$
  :
```

Figure 8. Vectorized computation of the element matrices and introduction into the general structure matrix

There is a lot of literature about vectorization of linear equation solvers available.[10,11] As it looks now, some kind of a renaissance of iterative approaches could take place, since they are easy vectorizable to a high degree. In the non-linear and time-dependent case they are especially effective, and the solution of the previous iteration provides reasonable approximations for the solution of the next step.

TEST RESULTS

Figures 9–11 show the results of the bench-marks in the form of speed-up factors, comparing the CPU time of the global vectorized program running on the Cyber 205 with the sequential program implemented on the Cyber 175 and Cyber 205, respectively, as a function of the spatial discretization.

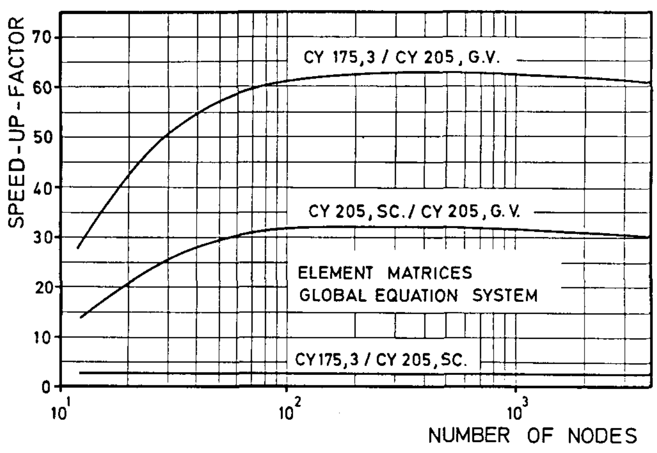


Figure 9. Speed-up factors: scalar (SC.) and global vectorized (G.V.) computation of the element matrices and assemblage of the structure matrix

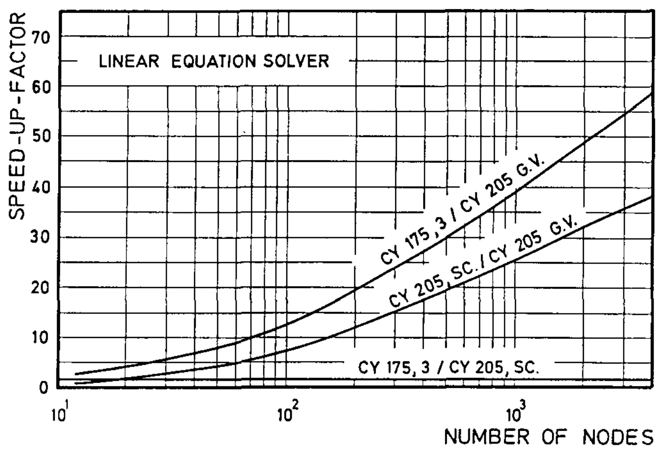


Figure 10. Speed-up factors: scalar (SC.) and global vectorized (G.V.) system equation solver

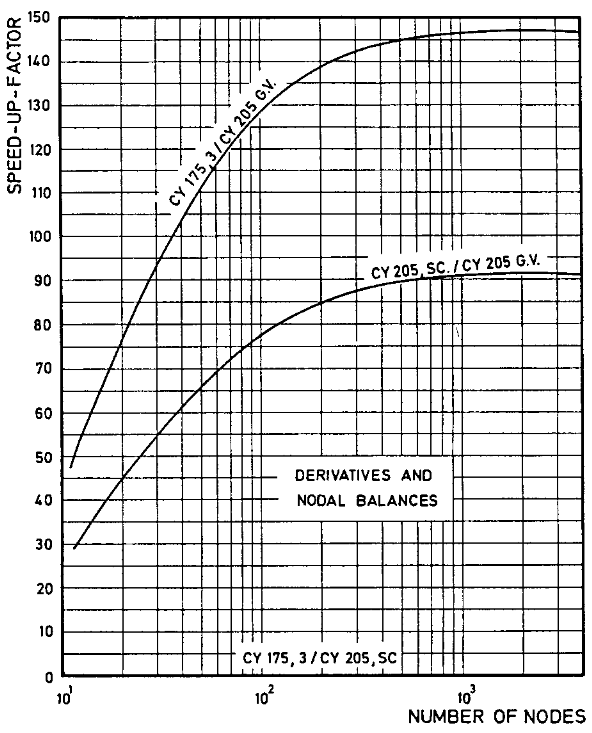


Figure 11. Speed-up factors: scalar (SC.) and global vectorized (G.V.) computation of the nodal balances and derivatives

First of all the results show again that scalar programs, developed and optimized for running on conventional serial computers, are not suitable to make use of the potentionally high performance of today's supercomputers. The minor speed-up is caused by the conventional hardware advance, only.

The speed-up of the Cyber 205 against the Cyber 175 (which has been for a comparatively long time one of the fastest computers available for the solution of scientific and engineering problems), is of special interest, when exploring the possibilities of the fourth generation vector computers in contrast to yesterday's supercomputers. But nevertheless this comparison includes also the results of conventional hardware progress, such as the shorter cycle times of the vector computer .

The results of the new parallel or vectorized hardware architecture in connection with new programming techniques, considering the new hardware characteristics become evident when comparing the CPU times of a scalar sequential program, vectorized at the local level by the compiler with the global vectorized program running on the same computer, showing remarkable speed-ups for all computationally massive parts of a finite element model.

CONCLUSION AND OUTLOOK

General theoretical considerations and results of extensive bench-marks show that transporting finite element models, developed and optimized to run on conventional sequential computers, to fourth generation supercomputers does not by far lead to the required speed-up to solve future computationally massive problems. The conventional program structures are not suited to make use of the potentionally great performance of today's supercomputers, and in extreme they may run even slower than when not vectorized at all.

A global change of the overall program logic, which means more or less a complete redesign, becomes necessary, considering the new hardware characteristics.

The global vectorization cannot be achieved by applying the FORTRAN 77 standard only. At least a minimum of additional intrinsic vector functions have to be used. Up to now, every manufacturer has supplied his own set of vector instructions, which prevents the portability of programs and the scientific exchange of experience and programs. Since the different vector instruction sets carry out basicaly the same vector operations, users should insist on establishing a vector language standard as soon as possible.

Vectorized programming needs more main memory, since on the one hand the definition of additional vectors and storage schemes become necessary and on the other hand slow input/output operations on background storage would not be in coincidence with extremely fast vector operations.

With a new generation of programs, developed and optimized to run on fourth generation vector computers, taking maximum advantage from the new advanced hardware architecture of vectorized or parallel processing, considerable speed-ups of more than an order of magnitude can be achieved and the numerical solution of new problem dimensions come into the reach of scientists and engineers.

ANNOUNCEMENTS

Funding for this study was provided by the German Research Foundation (DFG) under Ro 365/27. The authors wish to thank the Computer Centres of the University of Technology Aachen and the University of Bochum for the continuous co-operation and support.

NOTATION

$\vec{A}$	vector of element area
$\vec{H}$	vector of piezometric head
$\overrightarrow{HANF}$	vector of piezometric head of previous time step
$\overrightarrow{HE}$	vector of effective aquifer thickness
$\overrightarrow{HO}$	vector of caprock elevation
$\overrightarrow{HU}$	vector of bedrock elevation
$\vec{K}$	vector of element node number
$\overrightarrow{KF}$	vector of permeability
$\overrightarrow{MTYP}$	vector of material type
$\overrightarrow{RS}$	vector of right hand side
$\vec{S}$	vector of storativity (unconfined)
$\overrightarrow{SE}$	vector of effective storativity
$\overrightarrow{SO}$	vector of storativity (confined)
$\vec{T}$	vector of transmissivity

REFERENCES

1. W. Pelka, A. Peters and M. Vogt, 'Erste Erfahrungen mit dem Einsatz von Groeßtrechnern für mathematisch-numerische Grundwassermodelle, 6', *Bochumer Kolloquium* über Groeßtrechner und Anwendungen, Bochum, 1984.
2. R. Mares and R. Wojcieszynski, 'Vectorisieren in CYBER 200-Fortran', *Bochumer Schriften zur Parallelen Datenverarbeitung 3*, Rechenzentrum der Ruhr-Universitaet Bochum, 1983.

3. CDC, *Fortran 200 Version 1 Reference Manual 60480200 Rev. C*, Rechenzentrum Ruhr-Universitaet Bochum, 1984.
4. CRAY Research, Inc., *CRAY X-MP and CRAY-1 Computer Systems, Library Reference Manual SR 0014*, 1984.
5. A. J. Baker, *Finite Element Computational Fluid Mechanics*, McCrow Hill Book Comapny, 1983.
6. D. Withum, 'Elektronische Berechnung ebener und raeumlicher Sicker-und Grundwasserstroemungen durch beliebig berandete inhomogene anisotrope Mediên', *Mitteilungen des Instituts fuer Wasserwirtschaft und Landwirtschaftlichen Wasserbau der Technische Hochschule Hannover*, Heft 10, 1976.
7. K. H. Huebner, *The Finite Element Method for Engineers*, Wiley, New York, 1975.
8. O. C. Zienkiwicz, *The Finite Element Method in Engineering Science*, 2nd edn, McGraw Hill, London, 1971.
9. K. J. Bathe and E. L. Wilson, *Numerical Methods in Finite Element Analysis*, Prentice-Hall, 1976.
10. P. Concus, G. H. Golub and D. P. O'Leary, 'A generalized conjugate gradient method for the numerical solution of elliptical partial differential equations', *Sparse Matrix Computations*, Academic Press, 1976.
11. U. Schendel, *Einfuehrung in die parallele Numerik*, R. Oldenbourg Verlag, Muenchen Wien, 1981.