Invited Paper

Automatic Code Generation Method of DEQSOL (Differential EQuation SOlver Language)

CHISATO KONNO*, MICHIRU YAMABE*, MIYUKI SAJI*,
NOBUTOSHI SAGAWA*, YUKIO UMETANI, HIROYUKI HIRAYAMA**
and TADASHI OHTA**

DEQSOL is a high-level programming language specially designed to describe PDE problems in quite a natural way for numerical analyses. This language has two design targets. One is to enhance programming productivity by establishing a new architecture-independent language interface between numerical analysts and vector/parallel processors. The other is to automatically generate highly vectorizable FORTRAN codes from DEQSOL descriptions, thus realizing efficient execution.

The DEQSOL translator automatically generates highly vectorizable simulation codes using intrinsic parallelism remaining in DEQSOL descriptions. As a discretization method, both FDM and FEM are provided. The key techniques of the translator are the symbolic manipulation to discretize PDE and the code generation utilizing maximum DO-loops.

DEQSOL has been applied to over 30 practical problems, and the above targets are successfully attained. Productivity, when measured by the required source's lines-of-code, is improved by almost one order of magnitude over FORTRAN programming. Also, most of the generated FORTRAN codes have extremely high vectorization ratios (91%–96%) on the HITACHI S-810/20 vector processor.

1. Introduction

As indicated by the wide-spread use of supercomputers, demand for numerical simulations of physical phenomena has been increasing rapidly. Hardware used for such numerical simulation has made remarkable advances through innovations in LSI technology and computer architecture such as vector and parallel processors. However, the programming itself still remains at a relatively elementary level.

Simulation program development with such conventional languages as FORTRAN is faced with the following problems. One is that a long period of time is necessary to develop even a simple simulator. Another is that special knowledge of numerical analysis methods, like discretization, is needed. In addition, a specialized programming technique is required to exploit the performance of vector/parallel processors. Moreover, such programs are so lengthy and complicated that they can not be extended easily.

One approach for coping with these problems is the use of mathematical libraries or software packages designed for special application fields. However, there are distinct limitations in applicable fields and adopted numerical algorithms. Additionally, users can not fully control the numerical scheme in the details, and they often can not understand or utilize the complicated functions and interfaces.

To address these limitations, DEQSOL has been developed. DEQSOL (Differential EQuation SOlver Language) is a high-level programming language system specifically designed to describe PDE (Partial Differential Equation) problems in a natural way for numerical analyses. This system has two main purposes:

(1) To enhance programming productivity by establishing a new architecture-independent language interface between the numerical analyst and the computer.

(2) To generate highly vectorizable FORTRAN codes from DEQSOL descriptions using its intrinsic parallelism.

Several past or current simulation language systems such as SALEM[1], PDEL[2], and ELLPACK[3, 4] have been developed. In particular, ELLPACK seems to be a powerful system. ELLPACK has ample problem solving capabilities due to its extensive library, and its capabilities are still expanding in response to the demands for an interactive and distributed system environment (cf.[4]).

Recently, a program generator for fluid dynamics using a symbolic manipulator system has been reported (cf.[5]). PDE transformation facility using the boundary fitted coordinate transformation technique and its discretization facility are realized on MACSYMA[6], and furthermore, the program generator generates a

*Central Research Laboratory, Hitachi Ltd. Kokubunji, Tokyo 185, Japan
**Hitachi VLSI Engineering Ltd. Kodaira, Tokyo 187, Japan

calculation program by FORTRAN. However, the applicable field and descriptive capability of this system appear to be limited. Also performance is a problem because it is implemented upon a general purpose formula manipulator.

Compared with these systems, DEQSOL has the following significant advantages:

a) It posseses the capability to select and describe various numerical algorithms within the language.

b) It has the capability to describe complicated space regions with their automatic meshing facilities.

c) It provides both the Finite Difference Method (FDM) and Finite Element Method (FEM) as discretization facilities.

d) It generates FORTRAN simulation codes so that a high acceleration ratio can be realized with supercomputers.

Due to these features, DEQSOL is applicable to a wide range of practical and complex problems.

In this paper, an outline of DEQSOL, the automatic code generation method for supercomputers, and its evaluation through application to practical problems will be presented.

2. Outline of DEQSOL

First, an outline of DEQSOL is shown by using a simple example. A simple thermal diffusion problem and its description of a DEQSOL program are shown in Fig. 1. The DEQSOL program is divided into two definite parts, that is, the part defining structure of numerical model and the part describing numerical algorithm to solve problems.

Statement (1) declares the program name and statement (2) indicates the discretization method chosen by the user. Statements (3) to (10) specify the elements of the numerical model, namely, domain (DOM), time domain (TDOM), FDM meshes (MESH), unknown physical variables (VAR), known physical constants (CONST), subregions (REGION), initial condition (INIT), and boundary conditions (BOUND).

Between the 'SCHEME' statement (11) and the 'END SCHEME' statement (17), the numerical algorithm is described. To develop a numerical algorithm for this problem, time differentiation is replaced by forward difference and the left term is estimated according to either an explicit or implicit method. When the explicit method is selected, statement (14) is used, which assigns the evaluated value of the right terms to the left variable. Statements (12) to (17) represent a numerical algorithm by Euler's explicit method, where the statements between 'ITER' ation and 'END ITER' ation are executed until the specified condition is satisfied. For the implicit method, the 'SOLVE' statement is used, which solves the PDE for the indicated variable. If an implicit method such as backward Euler's method is selected, statement (14) should be replaced by the 'SOLVE' statement (19), This

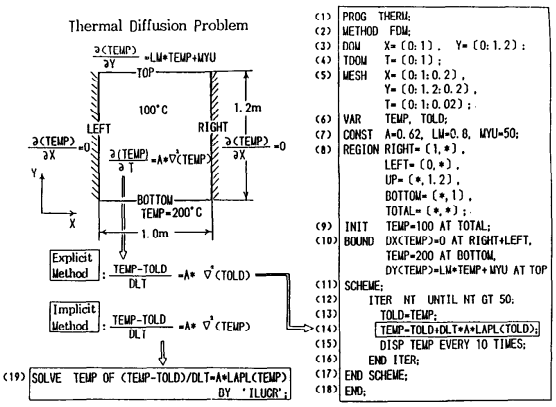

Fig. 1 A Simple Example of Description by DEQSOL.

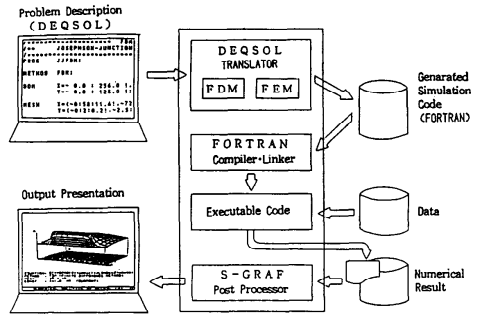

Fig. 2 Processing Flow of DEQSOL Program.

'SOLVE' statement is one of the key functions of DEQSOL. It is used to obtain the indicated variable value in the PDE which is linear with respect to the variable. The 'SOLVE' statement can be applied not only for transient problems, but also for static problems.

The structure of the DEQSOL system and its processing flow are shown in Fig. 2. The DEQSOL description is automatically translated into a FORTRAN simulation program by the DEQSOL translator. The DEQSOL translator has two discretization facilities, namely FDM and FEM. The generated FORTRAN program is executed on the ordinary route, and a large quantity of numerical result is analyzed by the post-processor S-GRAF.

The features of DEQSOL can be summarized as follows:

(1) It adopts the description form consisting of two distinctly separated parts. One part defines the structure of the numerical model and the other part describes the numerical algorithm. As a result, the numerical algorithm is easily modified independent of model part.

(2) In order to attain high productivity, readability

and extendability of program, concise styles and symbols commonly used by numerical analysts have been introduced. Namely,

a) New data types such as scalar, vector and tensor variables have been introduced to express various physical quantities.

b) Differential operators, such as first and second derivatives, gradient, divergence, rotation and Laplacian, have been introduced, so that field equations and conditions can be described in natural style.

(3) It is capable of describing complicated space regions and their automatic meshing schema.

(4) It can select and describe various numerical algorithms using the following functions in the scheme block:

a) Assignment statement with differential operators and the 'SOLVE' statement playing a basic role in updating or obtaining physical quantities.

b) 'ITER'ation, 'WHILE', 'IF-THEN-ELSE' block and linkage function to external modules to construct various numerical algorithms.

(5) Parallelism lurking in numerical algorithms can be expressed naturally by programmers without taking notice of it. That parallelism is utilized in generating vectorizable simulation programs by the DEQSOL translator.

At present, applicable fields of DEQSOL are the problems formulated by elliptic or parabolic PDEs on a 1-3 dimensional domain. As the automatic discretization method, both FDM and FEM are provided.

In the first example, the iteration block was used to control time steps. The iteration block and other control blocks also can be utilized for Newton Raphson's loop of non-linear PDE, for a successive scheme of simultaneous PDEs and so on. Other descriptions of numerical algorithms and more details of language specification can be found in the references ([7], [8], [9]).

3. DEQSOL Translator

In this section, how the DEQSOL translator generates simulation codes is described. The main concerns are the automatic discretization method and highly vectorizable code generation.

The parallelism existing in the numerical algorithm is classified into the following four types:

(A) Parallelism in discretization of region

This is a parallelism existing in the meshing process to represent the objective continuous region by finite nodes and elements. Parallel processing can be acheived for each coordinate or each subregion which is a unit of the automatic meshing process.

(B) Parallelism in discretization of PDE

This is a parallelism existing in the discretizing procedure inducing linear equations from PDE with FDM or FEM. It constructs the matrix by evaluating the coefficient elements and constant vector. There is mesh point-wize parallelism in this procedure.

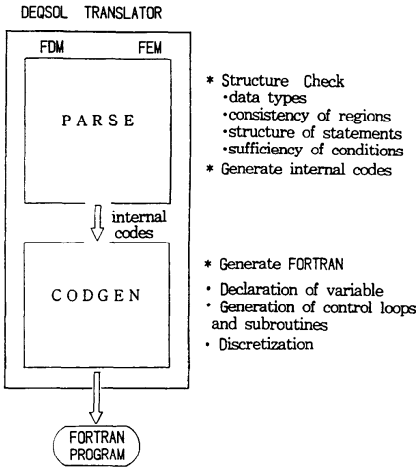

Fig. 3 DEQSOL Translator.

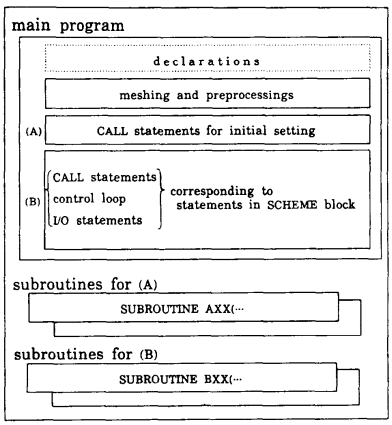

Fig. 4 Structure of Generated Codes.

(C) Parallelism for solving a linear equations system

This is a parallelism existing in solving the matrix induced by the above process. There are mesh point-wize, staggered mesh point-wize or other level parallelism according to matrix solution algorithms.

(D) Parallelism in scheme flow

This is an upper level parallelism in a numerical scheme algorithm. For example, according to the order of updating or refering to variables, parallel processing can be realized between PDEs or parts of the scheme.

All of these parallelisms remain natural in DEQSOL descriptions, and the translator (or compiler) can easily utilize them for code generation.

3.1 Outline of the translator and the generated code

As shown in Fig. 3, the DEQSOL translator consists of two main parts, namely PARSE and CODGEN (CODe GENerator). PARSE checks on the consistency

of regions, statement structure and sufficiency of intial or boundary conditions and finally generates intermediate codes. CODGEN generates FORTRAN codes from the intermediate codes, provides variable declarations needed in the program and generates control-loops and subroutines. Particularly with the 'SOLVE' statements, the DEQSOL translator discretizes the PDE according to the specified discretization method and generates codes that calculate the value of the matrix and constant vector elements in the discretized linear equations system.

The structure of the generated code is shown in Fig. 4. The codes consist of one main (control) program and many subroutines. At the head of the main program, there are declarations of variables defined by the 'VAR' or 'CONST' statements, declarations of the workarea for the indicated matrix solution library, and so on. The main procedure is divided into 3 parts. The first part provides meshing information and performs other preprocessings. The second part provides initial settings for known constants and initial conditions for variables corresponding to the 'CONST' or 'INIT' (initial condition) statements. The last part has essentially the same structure as the scheme block of DEQSOL, where the CALL statement, control statement or I/O statement is positioned corresponding to each executable statement of DEQSOL such as 'SOLVE', 'ITER' or 'PRINT' statements. The subroutines for each CALL statement come after the main program. However, one executable statement in the scheme block of DEQSOL often corresponds to several subroutines in order to improve compiling efficiency of FORTRAN as long as the parallelism in calculation is not destroyed.

At present, the main target machine for code generation by the DEQSOL translator is a pipeline vector processor, e.g. the HITAC S-810. Therefore, among the parallelisms lurking in the numerical algorithm, (A) to (C) are usable informations for parallel code generation. The key point is how to generate codes using maximal vectorizable DO loops. In DEQSOL, type (C) parallelism is attained by preparing highly vectorizable and efficient matrix solution libraries. Since most of the discretization of region is processed at translation time, the treatment of type (B) parallelism becomes the main objective of code generation.

In the following section, how to generate codes with type (B) parallelism is shown.

3.2 Code generation for FDM

The DEQSOL translator discretizes PDEs in the assignment statement or 'SOLVE' statement by following the built-in discretization rule. The elementary functions of the translator are symbolic manipulation of differential operators according to the discretization rule and the break down of region which extracts the maximal subregions where the unique discretized equation can be obtained.

First, the symbolic manipulation of differential

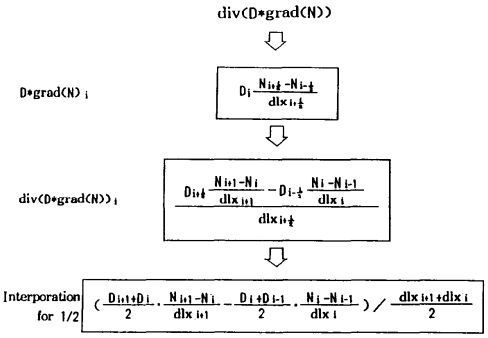

Fig. 5 Discretization Rule for Nested Differential Operators.

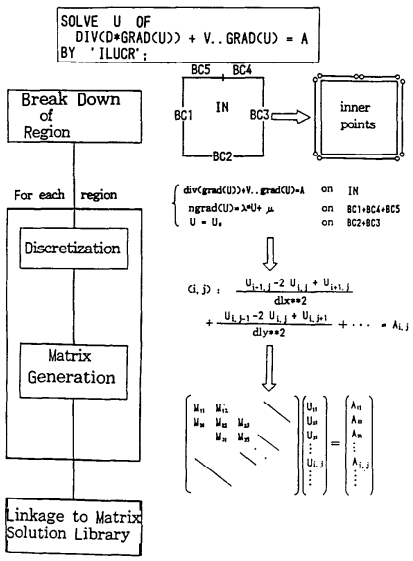

Fig. 6 Automatic Discretization Method.

operators in DEQSOL for FDM is shown. The key features of this method in DEQSOL are as follows:

*) Discretization is realized by the recursive procedure for differential operators, thus all PDEs can be managed.

*) Discretization is fulfilled by using the 1/2 index technique with the built-in discretization rule, which is equivalent to the finite volume method.

The typical flow of the discretization method for a diffusion term is shown in Fig. 5. Each term in a given PDE is extracted, and for each term, like div (D*grad (N)), the discretizing procedure is applied recursively from the inner differential operator. For example, as shown in Fig. 5, first, D*grad (N) is discretized, then discretization of divergence is applied to the discretized D*grad (N), and finally the procedure is finished by replacing the remaining 1/2 indices with the average of both sides of the index.

Another key function of automatic code generation is

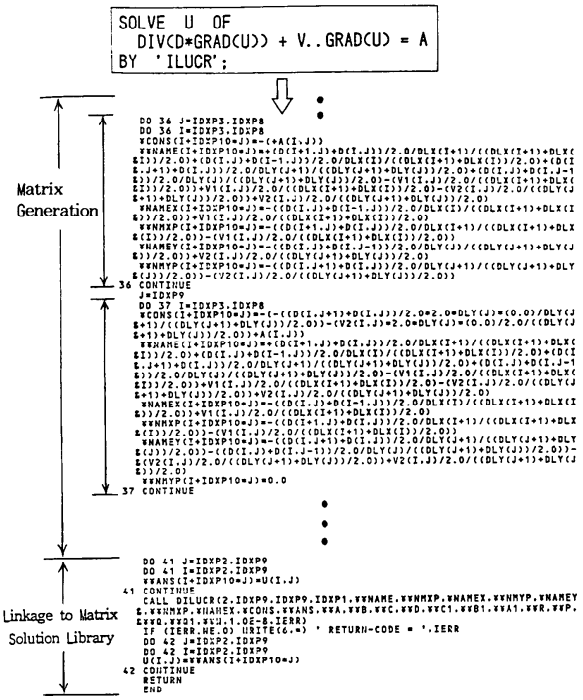

Fig. 7 Generated FORTRAN Code.

the break down of regions. This function is shown by using the example of the translation for the 'SOLVE' statement.

The translation method for the 'SOLVE' statement is described in Fig. 6. First, the region is broken down into several subregions according to the given PDE, boundary conditions, and discretization rule, so that each subregion is dominated by a unique equation and discretization rule. For example, the domain in the figure is broken down into 6 subregions: 5 boundary and 1 inner point subregions. Each subregion is a unit where the same pattern of linear equations is formed by discretization of every mesh point. As a result, the generated FORTRAN codes calculating portions of the total matrix and constant vector can be combined into a single DO-loop.

For each subregion, the DEQSOL translator discretizes the equation according to the discretization rule, as explained above. It then generates a part of the linear equations, and creates the FORTRAN codes that calculate the corresponding part of the total matrix and constant vector elements of the linear equation system. After the total matrix is completed, it is linked to the efficient matrix solution library prepared beforehand.

A portion of the generated FORTRAN codes consisting of the matrix generation and linkage to the matrix solution library is shown in Fig. 7. Each of the tightly nested DO loops in Fig. 7 are easily vectorized, because they contain no data dependencies (cf.[10]).

3.3 Code generation for FEM

The code generation method by FEM is essentially similar to that of FDM. However, the symbolic manipulation is based on the Galerkin method. At first, an outline of this method is shown by letting

$$\begin{cases} -\nabla\cdot(T\nabla A)+\beta * A+F=0 & \\ A=AO & AT\ \partial\Omega_1 \\ n\cdot(T\nabla A)=\lambda * A+\mu & AT\ \partial\Omega_2 \\ & (\partial\Omega=\partial\Omega_1+\partial\Omega_2) \end{cases} \tag{E1}$$

be a sample target equation.

In the finite element method, the objective unknown quantity A is approximated by a linear combination of the basis function φ_j which has a unit value at the j-th node. In place of the unknown quantity A coefficients $\{a_j\}$ are derived as the values of A at nodes $\{N_j\}$. In formula (E2), NODE denotes the number of nodes in region Ω to be simulated.

$$A=\sum_{j}^{\text{NODE}} a_j\varphi_j \tag{E2}$$

To determine the coefficients $\{a_j\}$, the PDE (E1) is multiplied by φ_i and the resultant function is integrated in the simulation region Ω, and reduced to the following equation.

$$-\int_\Omega(\nabla\cdot(T\nabla A))\varphi_i+\int_\Omega \beta A\varphi_i+\int_\Omega F\varphi_i=0 \tag{E3}$$

Here, the following equivalent relational expression (E4) is introduced.

$$-\int_\Omega(\nabla\cdot(T\nabla A))\varphi_i=\int_\omega(T\nabla A)\cdot\nabla\varphi_i-\int_{\partial\Omega} n\cdot(T\nabla A)\varphi_i \tag{E4}$$

where $\partial\Omega$ indicates the boundary of region Ω. On the boundary $\partial\Omega_2$, expression (E5) holds as follows from the boundary condition of equation (E1).

$$-\int_\Omega(\nabla\cdot(T\nabla A))\varphi_i=\int_\Omega(T\nabla A)\cdot\nabla\varphi_i-\int_{\partial\Omega_2}\lambda A\varphi_i-\int_{\partial\Omega_2}\mu\varphi_i \tag{E5}$$

By substituting equations (E5) and (E2) for (E3), the following equation (E6) is derived. Equation (E6) holds when i indicates a node number of a node on the boundary $\partial\Omega_2$ or a node inside the region Ω.

$$\sum_{j}^{\text{NODE}}\left\{\int_\Omega(T\nabla\varphi_j)\cdot\nabla\varphi_i+\int_\Omega\beta\varphi_j\varphi_i-\int_{\partial\Omega_2}\lambda\varphi_j\varphi_i\right\}a_j$$
$$=-\int_\Omega F\varphi_i+\int_{\partial\Omega_2}\mu\varphi_i \tag{E6}$$

On the other hand, expression (E7) holds on the boundary $\partial\Omega_1$.

$$a_j=AO_j \tag{E7}$$

By using the equations (E6) and (E7), simultaneous linear equations with respect to coefficients $\{a_j\}$ are obtained. The equations can be represented as a matrix equation (E8) as follows:

$$[K_{ij}][a_j]=[d_j] \tag{E8}$$

where;

$$K_{ij}=\int_{\Omega}(T\nabla\varphi_j)\cdot\nabla\varphi_i+\int_{\Omega}\beta\varphi_j\varphi_i-\int_{\partial\Omega_2}\lambda\varphi_j\varphi_i$$

$$d_j=-\int_{\Omega}F\varphi_i+\int_{\partial\Omega_2}\mu\varphi_i$$

when i denotes a node number of a node on the boundary $\partial\Omega_2$ or a node inside the region Ω. Furthermore, the integrals of basis function are replaced by the formulae refering to the coordinate value of nodes.

Consequently, by solving the simultaneous linear equations (E8), the value of coefficients $\{a_j\}$ ($j=1, 2, \ldots,$ NODE) can be derived. The process of forming the simultaneous linear equations (E8) from the original equation (E1) is called discretization, which is the main task of the DEQSOL translator.

The symbolic manipulation explained above has been realized in the DEQSOL translator due to the following key functions:

a) Generation of the integral form by the weighted residual procedure, corresponding to the (E3) process.

b) Integral transformation using equivalent rules such as divergence theorem and partial differentiation formulae. This corresponds to the (E4) and (E5) processes.

c) Manipulation of numerical integration formulae for the basis function, corresponding to the (E8) process.

The basic idea for the code generation is similar to FDM. Total matrix K_{ij} consists of the contribution from the region integral terms and the boundary integral terms. The region integral terms are related to all nodes. On the other hand, the boundary integral terms are related to only nodes on the boundary and the modification of (E7) is needed only for nodes on the Direchlet boundary. Therefore, the contribution of region integral terms can be calculated by a single DO-loop for all nodes and the contribution of boundary integral terms can be calculated by the other DO-loops for nodes on boundary regions according to the different boundary conditions.

A part of the generated FORTRAN code computing the matrix elements and constant vector elements is shown in Fig. 8. RCOEF is a two dimensional array that holds non-zero values of the matrix elements. CONS is a one-dimensional array that holds constant vector values. This DO-loop corresponds to the calculation of the contribution of the region integrals. Vector length is the node number. The innermost DO loop is efficiently executed by the S-810 supercomputer with the use of control vector and list vector facility (cf.[11]).

```
      :
      DO 54 I3=1,2
      DO 54 I2=1,BAND
      DO 54 I1=1,NODE
      IF (ELM(I1,I2,I3).NE.0) THEN
      IF (ELM(I1,I2,I3).EQ.IADATA(ELMO(I1,I2,I3),1)) N=1
      IF (ELM(I1,I2,I3).EQ.IADATA(ELMO(I1,I2,I3),2)) N=2
      IF (ELM(I1,I2,I3).EQ.IADATA(ELMO(I1,I2,I3),3)) N=3
      K=IADATA(ELMO(I1,I2,I3),N)
      I=IADATA(ELMO(I1,I2,I3),MOD(N,3)+1)
       J=IADATA(ELMO(I1,I2,I3),MOD(N+1,3)+1)
       BI=Y(J)-Y(K)
       CI=X(K)-X(J)
       BJ=Y(K)-Y(I)
       CJ=X(I)-X(K)
       BK=Y(I)-Y(J)
       CK=X(J)-X(I)
       RCOEF(I1,I2)=RCOEF(I1,I2)+(((BJ/(2*ARE(I1,I2,I3)))*((BI/(2*ARE(I1
     &,I2,I3)))+((CJ/(2*ARE(I1,I2,I3)))*((CI/(2*ARE(I1,I2,I3))))*(-AR
     &E(I1,I2,I3))-(((KK(I)*1.0/LAMD**2)+(KK(J)*1.0/LAMD**2)+(KK(K)*1.0/
     &LAMD**2))/3)*ARE(I1,I2,I3)/12.0
       CONS(I1)=CONS(I1)-(-((KK(I)+KK(J)+KK(K))/3*(-(XX(I)+XX(J)+XX(K))/3
     &))*ARE(I1,I2,I3)/3.0)
       ENDIF
54    CONTINUE
      :
```

Fig. 8 Example of FORTRAN Code Generated by DEQSOL Translator for the Solve Statement (FEM).

4. Evaluation

In this section, an evaluation of DEQSOL, such as programming productivity, and vectorization ratio of generated codes is given through its application to practical problems.

DEQSOL and its processing system have been under trial use for 5 years and applied to over 30 practical problems. Applied fields have included thermal conduction problems, convection and diffusion problems for LSI manufacturing processes, electric or magnetic field analysis, fluid analysis, etc. In the view point of numerical algorithms, explicit and implicit methods for time-dependent problems, the Newton-Raphson's method for non-linear PDE, the successive method for simultaneous PDEs and also the successive method for integro-differential equations and other schemes have been tried.

A few evaluations are shown in Table 1. The descriptive efficiency ratio of DEQSOL (lines-of-code) and the runtime efficiency of the generated FORTRAN code (vectorization ratio, acceleration ratio using vector processor) for each practical problem are illustrated in Table 1. Here, acceleration ratio refers to the speed-up rate for the generated FORTRAN codes executed on vector processor S-810 against that for the scalar processor of the same machine.

(1) Productivity

Productivity, when measured by the required source's lines-of-code, has been improved by almost one order of magnitude over FORTRAN programming. Here, in Table 1, FORTRAN's lines-of-code for 2 problems marked with (*), that is CVD and JJD, are not for generated codes, but for that of existing FORTRAN programs.

(2) Vectorization ratio

The vectorization ratio in Table 1 are measured by the HITAC S-810/20. Most of the generated FORTRAN codes have extremely high vectorization ratio (91.0–96.4%). When they are executed on the vector processor, the executions are accelerated 3.8 and 9.4 times faster than by scalar operations. These results are

Table 1 Evaluation of DEQSOL (FDM/FEM) (Productivity & Vectorization Ratio).

Program Name		CVD	PCAD	WEL	JJD	MDH	EBT
Discretization Method (Dim.)		FDM (3)	FDM (2)	FDM (2)	FEM (2)	FEM (2)	FEM (2)
Lines-of-code	DEQSOL	127	79	67	96	904	239
	Generated FORTRAN (Ratio)	1,361 (*) (10.7)	1,312 (16.6)	1,091 (16.3)	1,078 (*) (11.2)	11,558 (12.8)	2,476 (10.4)
Vectorization Ratio (S-810) [%]		91.0	96.4	94.1	92.7	93.9	93.7
Acceleration by Vector Processor (S-810)		3.8	8.2	9.4	5.2	7.7	5.5

CVD: Flow Analysis of Chemical Vapour Deposition
PCAD: Impurity Distribution Analysis in LSI Process CAD
WEL: Device Simulation of WEL Layer in LSI
JJD: Magnetic Field Analysis of Josephson Junction Device
MDH: Magnetic Field Analysis of Disc Head
EBT: Field Potential Analysis of Electron Beam Tube

attained by making discretization codes and linear equation solving processes highly vectorizable, since they are the bulk of the calculations.

5. Conclusion

Language features, the automatic code generation method and an evaluation of DEQSOL have been shown through application to practical problems.

New data types, differential operators and control functions have been introduced to DEQSOL. Through these functions, advanced numerical algorithms can be described in a compact form retaining intrinsic parallelism of the algorithm.

Automatic code generation of the DEQSOL translator have been achieved through both FDM and FEM. The key functions of the automatic code generation method are symbolic manipulation of the discretization process and the control function of subregions (e.g. the break down of region in FDM case) so that as many calculations as possible can be bundled up by a single DO-loop to facilitate vectorization.

DEQSOL have been applied to a wide range of problems. Simulation program productivity has been improved by almost one order of magnitude. Also, most of the generated FORTRAN codes have extremely high vectorization ratios (91–96%) on the HITAC S-810/20.

Acknowledgements

The authors would like to express their appreciation to Dr. Sakae Takahashi of Hitachi Ltd. for his helpful technical discussions. Thanks are also due to Dr. Tsuneyo Chiba, Dr. Hisashi Horikoshi and Dr. Yasutsugu Takeda of Hitachi Ltd. for their continuous encouragements.

References

1. MORRIS, S. M. and SCIESSER, W. E. SALEM—A Programming System for the Simulation of Systems Described by Partial Differential Equations, *Proc. Fall Joint Computer Conference* **33** (1968), 353–357.
2. CARDENAS, A. F. and KARPLUS, W. J. PDEL—A Language for Partial Differential Equations, *Com. ACM* (March 1970), 184–191.
3. RICE, J. R. and BOISVERT, R. F. Solving Elliptic Problem Using ELLPACK, CSD-TR414, *Computer Science Department, Purdue University*, (September 1982 Revised May, 1983).
4. RICE, J. R. ELLPACK: An Evolving Problems Solving Environment *Proc. of IFIP* WG 2.5 *Working Conference on Problem Solving Environments for Scientific Computing* (1985), to be published.
5. STEINBERG, S. and ROACHE, P. J. Symbolic Manipulation and Computational Fluid Dynamics, *J. Comput. Phys.* **57** (1985), 251–284.
6. MACSYMA Reference Manual, The MATHLAB Group, *Laboratory for Computer Science, MIT*, Ver **10** (1983).
7. UMETANI, Y. *et al.* Numerical Simulation Language DEQSOL (Japanese), *Transaction of Information Processing Society of Japan* **26** (1985), 168–180.
8. KONNO, C. *et al.* Advanced Implicit Solution Function of DEQSOL and Its Evaluation, *Proc. of Fall Joint Computer Conference* **86** (1986), 1026–1033.
9. KONNO, C. *et al.* A High Level Programming Language for Numerical Simulation: DEQSOL, *IEEE Tokyo Section, Denshi Tokyo* **25** (1986), 50–53.
10. TAKANUKI, R. *et al.* Some Compiling Algorithms for an Array Processor, *Proc. of 3rd USA-Japan Computer Conference*, (1978), 273–279.
11. YASUMURA, M. *et al.* Compiling Algorithms and Techniques for S-810 Vector Processor, the 1984 International Conference on Parallel Processing, *IEEE Press, New York* (1984), 285–290.

(Received August 26, 1987)