C.E.A.
Etablissement : C.E. de SACLAY
Direction des Réacteurs Nucléaires
Département de Mécanique & de Technologie

Date LE 20/09/94
Réf. : DMT/SERMA/LETR/94/100

CEA-DMT

DEMANDE D'AUTORISATION DE COMMUNICATION OU DE PUBLICATION

PUBLICATION - TITRE :

Auteur (s) :
Nature de la Publication :
Editeurs (CEA, AIEA, OCDE,...) :

COMMUNICATION - TITRE : SOLVING LINEAR SYSTEMS IN FLICA 4, THERMOHYDRAULIC CODE FOR 3-D
Auteur (s) : TRANSIENT COMPUTATIONS DE G.ALLAIRE

Titre de la Conférence : ANS INTERNTIONAL CONFERENCE ON MATHEMATICAL AND COMPUTATIONS REACTORS PHYSISCS AND ENVIRONNEMENT ANALYSES

Lieu et Date : PORTLANC OREGON USA LE 30 AVRIL AU 04 MAI 1995

ORGANISATEURS : ANS

ARRIVEE LE 14 MAR 1995 S3DS/SPR

ARRIVÉE - CIRST
13 OCT 94 | 005889
Circ. : LM | CIt : NC

Date de remise des textes :
Commentaires :

Ce texte a-t-il été déjà publié ? :
Si OUI : Référence de la publication antérieure :
Cette publication contient-elle, à votre avis, des informations brevetables OUI NON

DATE DE LA DEMANDE : 20/09/94

Chef de Laboratoire : P. RAYMOND

Chef de Service : J.B. THOMAS

DEMANDE D'AVIS (éventuellement) :

a) - C.P.I./Saclay	OUI	NON
b) - Chargé de Mission pour les affaires industrielles	OUI	NON
c) - SYFRA	OUI	NON
d) - Partenaires concernés (EDF, FRA, Dpts de la DRN) : (joindre photocopie)	Date de la Demande :	
e) - Autres unités opérationnelles : (joindre photocopie)	Date de la Demande :	
f) - Autres avis demandés par le Chef du D.M.T. :		

Avis du : (le cas échéant)

Date :
les avis sont à envoyer au DMT/DIR

Date d' Arrivée au D.M.T. 22-9-94

Décision D.M.T. n° 208 Accord

Date 23-9-94

Le Chef du Département

P.J. : Un texte complet -
Copie : DMT/DOC
~~DMT/EA~~
Nota : Copie autorisation + Résumé + Texte à I.N.S.T.N./MIST/CIRST.

SOLVING LINEAR SYSTEMS IN FLICA-4.
A THERMOHYDRAULIC CODE FOR 3-D TRANSIENT COMPUTATIONS.

Grégoire ALLAIRE
Commissariat à l'Energie Atomique
DRN/DMT/SERMA. C.E. Saclay
91191 Gif sur Yvette. France

Abstract

FLICA-4 is a thermohydraulic code developed at the French Atomic Energy Commission (CEA) for computing three-dimensional, transient or steady-state, two-phase flows in nuclear reactor cores. It aimed at solving efficiently both, small size problems (around 100 mesh cells), and large ones (more than 100,000), on either standard workstations or vector super-computers. As is well-known for time implicit codes, the largest time and memory consuming part of FLICA-4 is the routine dedicated to solve the linear system (the size of which is of the order of the number of cells). Therefore, the efficiency of the code is crucially influenced by the optimization of the algorithms used in assembling and solving linear systems. The goal of this paper is to describe how this is done in FLICA-4, and to report performances obtained in typical test cases.

1 A brief description of FLICA-4.

FLICA-4 is a computer code devoted to steady state and transient thermal-hydraulic analysis of nuclear reactor cores. The two-phase flow model of FLICA-4 is the so-called drift-flux model (see e.g. [12]) for a mixture of water and steam, which is based on four conservation laws :

1. a mixture (or total) mass balance equation,
2. a mixture momentum balance equation,
3. a mixture energy balance equation,
4. a phasic mass balance equation for the saturated phase.

The two phases are assumed to have the same pressure, and their velocities are related by an algebraic "drift" relation. Thermal conduction and viscous effects are taken into account, and various physical terms or closure relationships are also included in the model such as subcooled boiling, bulk condensation, turbulent diffusion, wall friction, grid effects... (see [9] for details).

These four conservation laws give rise to six scalar equations (the momentum has three components) which are solved by using a finite volume method.

Therefore, there are six unknown quantities per mesh cell, and their variations at each time step is a balance of the discretized fluxes between adjacent cells. The precise computation of these discretized fluxes is based on an approximate Rieman solver of Roe's type [5], developed for two-phase flow in [8]. As is well-known Roe's numerical scheme is based on an adequate linearization of the equations and requires the knowledge of the Jacobian matrix (i.e. the derivative of the fluxes, see [5]). Therefore, it is easy to build a linearized time-implicit version of this scheme. More precisely, for a 1-D inviscid flow with no source terms (for simplicity) the model equations are of the form

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0.$$

Upon spatial discretization, they become

$$\frac{\partial U}{\partial t} + A(U)U = 0,$$

where the components of the vector U are averages of the function $u(x)$ in the cells, and $A(U)$ is the Jacobian matrix discretized by Roe's scheme. Denoting by U_n the value of of U at the time $n\Delta t$ (where Δt is a given time step), a linearized implicit scheme is

$$\frac{U_{n+1} - U_n}{\Delta t} + A(U_n)U_{n+1} = 0. \tag{1}$$

At each time step, the updated solution U_{n+1} is obtain from the previous solution U_n by solving the linear system (1). The corresponding matrix to invert is

$$\mathcal{A} = \frac{1}{\Delta t} Id + A(U_n), \tag{2}$$

which is non-symmetric, and usually large and sparse. Its conditioning is very good for small time-steps, and get worse as Δt increases.

The goal of this paper is to describe how to solve efficiently the linear system (1) in terms of CPU time and memory requirement to store the matrix $\mathcal{A}$. Furthermore, the FLICA-4 code has to handle small size problems, as well as large ones, and to operate either on standard workstations or on vector super-computers. As a consequence, different methods and optimizations are presented in the sequel to take into account this variety.

2 Matrix structures and storages.

As is well-known, the structure of the Jacobian matrix $\mathcal{A}$ defined by (2) depends on the geometry of the mesh. FLICA-4 deals with semi-structured meshes in the following sense : a 3-D mesh is obtained by translation in the z-direction of any unstructured $x - y$ mesh. Therefore, the mesh section is the same at

each level z. Since each level interacts only with himself and its two upper and lower neighbours, this yields a bloc tridiagonal structure for $\mathcal{A}$, where each bloc corresponds to a small matrix of the size of the $x-y$ mesh (see Figure 1).

$$\mathcal{A} = \begin{pmatrix} D_1 & U_1 & & & & \\ L_2 & D_2 & U_2 & & 0 & \\ & L_3 & D_3 & U_3 & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & L_{n-1} & D_{n-1} & U_{n-1} \\ & & & & L_n & D_n \end{pmatrix}$$

Figure 1: *Bloc tridiagonal matrix.*

FLICA-4 usually works with this bloc matrix. This yields a good granularity of the data. Indeed, FLICA-4 uses the ESOPE library [11] which manages a virtual memory space on a disk. Therefore, the code can run with only a few blocs in real memory, while the other ones are written and available on a disk. This allows to run large problems for which the computer memory is too small (at the expense of an increase in the number of I/O's). However, for small problems, FLICA-4 switches to a global band storage of the matrix $\mathcal{A}$ (see the next section).

According to the type of target computer, the blocs L_i, D_i, U_i are stored : in band form to favor vectorization, or in sparse form to optimize the storage on scalar workstations. The sparse structure reflects the connections of the cells in the section $x-y$ mesh, and is thus the same for all levels $1 \leq i \leq n$. This minimizes the overhead of the book-keeping due to the sparse storage (see [1] for details). Of course, the indirect addressing caused by the sparse storage disables partly the vectorization, which means that this type of storage is recommended only for scalar computers. Figure 2 gives the necessary amount of storage memory in different cases arising from standard core computations (*the number of cells is decomposed in the number of levels times the number of* cells in the $x-y$ section).

number of cells	global band	bloc band	bloc sparse
31*26 = 806	1.8	0.5	0.6
31*157 = 4,867	60.6	8.3	4.4
62*628 = 38,936	1847.2	129.6	37.0
93*1,413 = 131,409	13803.4	650.4	127.4

Figure 2: *Storage requirement of the matrix $\mathcal{A}$ (in MegaWords).*

As is easily seen, the global band storage (for a Gauss decomposition) can only be used for small problems. The two bloc storages are used with a conjugate gradient method and they minimize the required memory. Note that the size of system is 6 times the number of cells (due to the 6 equations to solve), and that we already included the preconditioning storage in the above figures.

3 Direct method : Gauss decomposition.

For moderate size problems, a direct method as the Gauss (or LU) decomposition is perfectly adequate to solve the linear system (1). It is robust and gives an exact result. (Incidentally, the definition of "moderate" depends on the available memory capacity of a considered computer.) Gauss decomposition is also well-suited for vectorization. Its only inconvenient is its high requirement of memory storage. To cope with this problem, Gauss decomposition has been implemented in two ways. First, for small systems, the global matrix is stored in band form in a single array. Second, for larger systems the matrix is stored in independent blocs, and the Gauss decomposition is performed on this bloc form (this allows to write part of the matrix on the disk and minimize the actual storage in the real memory).

Figures 3 and 4 report on CPU time measures for the two smallest matrix considered in the previous section. The two steps of the algorithm are measured separately : first the Gauss decomposition, and then the solving of the resulting triangular system. Remark that for the 31*157 mesh, the bloc Gauss decomposition is possible on a workstation thanks to the ESOPE library which emulates a virtual memory on the internal disk of the machine. Megaflops counts on the Cray C90 have been obtained with the Perfview utility (a single processor is used). Computation on the SUN SS10 have been performed with the standard Fortran compiler of SunOS 4.1.3.

computer		global LU decomposition	global LU solution	bloc LU decomposition	bloc LU solution
CRAY C90	CPU s.	1.2	0.02	1.4	0.03
	Mflops	273.	234.	288.	167.
SUN SS10	CPU s.	51.	0.4	55.	0.45
	Mflops	6.4	11.7	7.3	11.1

Figure 3: *CPU measures for the 31*26 mesh.*

computer		global LU decomposition	global LU solution	bloc LU decomposition	bloc LU solution
CRAY C90	CPU s.	146.6	0.27	176.5	0.32
	Mflops	418.9	454.9	438.7	403.4
SUN SS10	CPU s.	***	***	14333.	92.7
	Mflops	***	***	5.4	1.39

Figure 4: *CPU measures for the 31*157 mesh.*

4 Iterative method : conjugate gradient.

For large problems, Gauss decomposition is inefficient, or even impossible, due to large CPU and memory requirement. Iterative methods provide an alternative, and among them the preconditioned conjugate gradient is the method of choice. Different variants of this algorithm are implemented in FLICA-4 : the conjugate gradient squared (CGS) of [7], its stabilized version (Bi-CGSTAB) of [10], and the generalized minimal residual (GMRES) of [6]. In most cases, GMRES performs very well and is 10 or 20% better than the two other ones. As is well-known, the choice of the preconditioning is crucial to achieve a fast convergence. To stay in the framework of a bloc structure of the matrix, we chose the SSOR preconditioning (which is derived from the famous SSOR method of [13], see e.g. [4]) coupled with Eisenstat's trick [3]. Denoting respectively by L, D, U the matrices of the lower diagonal, diagonal, and upper diagonal blocs, the system to solve is

$$Ax = b, \text{ with } \qquad D + U), \tag{3}$$

where b is a given right hand side. The main idea of the SSOR preconditioning is that

$$(L + D + U)^{-1} \approx (D + U)^{-1} D (L + D)^{-1}. \tag{4}$$

Then, multiplying the global matrix by the approximate inverse in (4), the SSOR preconditioning amounts to replace system (3) by a better conditioned system :

$$\tilde{A}\tilde{x} = \tilde{b}, \tag{5}$$

$$\text{with } \tilde{A} = (I + D^{-1}L)^{-1} + (I + D^{-1}U)^{-1} - (I + D^{-1}L)^{-1}(I + D^{-1}U)^{-1},$$

where $\tilde{b} = (L + D)^{-1} b$ and $\tilde{x} = (I + D^{-1}U)x$. Remark that this type of preconditioning requires only to compute the Gauss decomposition of the diagonal blocs, and to perform multiple solves of triangular bloc matrices. Compared to a simple bloc diagonal preconditioning, it requires about ten times iterations less to converge. Its efficiency is comparable to that of a global incomplete (of degree 0) LU factorization (see e.g. [4]) which has the inconvenient of not being a bloc algorithm.

The SSOR preconditioning can be implemented either with a band or a sparse storage of the blocs. To decrease further the size of the storage, FLICA-4 features also a variant where a mere incomplete LU factorization of degree 0 (i.e. with no fill-in) of the diagonal blocs is performed. Although we now have to keep in memory both the diagonal blocs and their incomplete factorization, the gain due to the no-fill-in is important. Of course, the resulting preconditioned system is not as simple and well conditioned as (5) (for details, see [1]). In the sequel, we will refer to this preconditioning as the ILU(0) SSOR preconditioning. Figure 5 and 6 below display CPU measurements for the SSOR-preconditioned GMRES algorithm. Convergence was detected when the error was reduced from

a factor of 10^5. For the 31*157 mesh, it requires 14 iterations for the band and sparse GMRES, and 21 for the sparse ILU(0) GMRES. For the 62*628 mesh, it requires 4 iterations for the band and sparse GMRES, and 24 for the sparse ILU(0) GMRES.

		band		sparse		sparse ILU(0)	
computer		preco.	iter.	preco.	iter.	preco.	iter.
C90	CPU s.	2.66	4.71	11.3	19.9	1.19	14.2
	Mflops	214.	202.	47.7	23.1	53.7	32.9
SS10	CPU s.	86.8	665.6	58.2	93.9	5.37	87.2
	Mflops	6.6	1.43	9.2	4.9	11.9	5.3

Figure 5: *GMRES for the 31*157 mesh.*

		band		sparse		sparse ILU(0)	
computer		preco.	iter.	preco.	iter.	preco.	iter.
C90	CPU s.	61.2	79.0	345.2	151.4	19.9	129.6
	Mflops	283.	306.	42.2	16.3	54.1	33.1
SS10	CPU s.	***	***	***	***	221.	955.
	Mflops	***	***	***	***	4.9	4.4

Figure 6: *GMRES for the 62*628 mesh.*

Figures 5 and 6 show that for large problems conjugate gradient methods are very competitive, in terms of both, CPU time, and memory requirement. Depending on the conditioning and the size of the matrix, a sparse or a band storage of the blocs is preferable. Remark however that a sparse algorithm destroys the vectorization as is obvious from the Megaflops count on the Cray C90. Of course, one should favor a band algorithm on vector computers, and a sparse algorithm on scalar workstations.

5 Conclusion.

FLICA-4 features many different algorithms to solve linear systems, but how to choose the best one for a given problem ? We conclude this paper by giving a brief user's guide for solving linear systems. In the first place there is a big difference in steady-state or transient computations. For transient simulations, the time-step is usually small (yielding a good conditioning of the matrix) and the Jacobian matrix must be updated at each time-step. Thus, apart from very small problems (as the 26*31 mesh of Figure 3), one should prefer an iterative conjugate gradient method with a sparse storage on scalar computers and a band one on vector machines. On the other hand, for steady-state computations, FLICA-4 performs a pseudo-transient with large time-steps in order to reach as fast as possible the steady solution. Thus, the matrix is usually not so-well conditioned. Furthermore, the details of the pseudo-transient are of no

importance, and it turns out that the Jacobian matrix has not to be necessarily updated at each time-step to achieve a good convergence. For example, the Jacobian matrix can be stored and updated only every 100 time steps. This process favors a direct method since the more costly part of the Gauss decomposition can now be spared on 99 steps. This is the main reason why direct methods are not completely outperformed by iterative one.

Finally, when memory space is the ultimate criterion (as, for example, in the last 93*1413 mesh in Figure 2), one has to rely on an iterative method with a sparse storage. Alternatively, parallel computers with a distributed memory provide a way of handling such very large systems. We are currently implementing a decomposition domain method in FLICA-4 which is well-suited for parallelism. Of course, in the context of an iterative method, the SSOR preconditioning (which is actually sequential) has to be modified (see [2] for details). However, parallel and distributed computing is a very promising way for solving large linear systems.

Acknowledgements. The FLICA-4 computer code has been developed at the Service des Réacteurs et de Mathématiques Appliquées (SERMA), Département de Mécanique et de Technologie (DMT), Direction des Réacteurs Nucléaires (DRN) of the Commissariat à l'Energie Atomique (CEA), by a team including G. Allaire, G. Boudsocq, D. Caruge, Th. de Gramont, P. Raymond, and I. Toumi.

References

[1] G. Allaire, *FLICA-IV version 1.0 : manuel de référence des modules de résolution des systèmes linéaires*, Internal report DMT 92/536 (in French), Direction des Réacteurs Nucléaires CEA (1992).

[2] G. Allaire, *Préconditionnement parallèle pour une matrice tridiagonale par blocs*, Internal report DMT 93/615 (in French), Direction des Réacteurs Nucléaires CEA (1993).

[3] S. Eisenstat, *Efficient implementation of a class of preconditioned conjugate gradient methods*, SIAM J. Sci. Stat. Comput. 2, pp.1-4 (1981).

[4] J. Ortega, *Introduction to parallel and vector solution of linear systems*, Plenum Press, New York (1988).

[5] P. Roe, *Approximate Riemann solvers, parameter vectors, and difference schemes*, J. Comput. Phys. 43, pp.357-372 (1981).

[6] Y. Saad and M. Schultz, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput. 7, pp.856-869 (1986).

[7] P. Sonneveld. *CGS: a fast Lanczos-type solver for nonsymmetric linear systems.* SIAM J. Sci. Stat. Comput. 10. pp.36-52 (1989).

[8] I. Toumi. *A weak formulation of Roe's approximate Rieman solver.* J. Comput. Phys.

[9] I. Toumi. D. Caruge, and P. Raymond. *An implicit second order numerical method for 3-D two-phase flow calculations.* in Proceedings of the 1994 Topical Meeting on Advances in Reactor Physics. April 11-15, Knoxville (1994).

[10] H. 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, pp.631-644 (1992).

[11] P. Verpeaux, *Manuel d'utilisation ESOPE (Version 10.0) et GEMAT (Version 10.0),* Internal report DMT 93/617 (in French), Direction des Réacteurs Nucléaires CEA (1993).

[12] G. Wallis, *One-dimensional two-phase flow,* McGraw-Hill, New York (1969).

[13] D.M. Young, *Iterative solution of large linear systems,* Academic Press (1971).