PRELIMINARY INVESTIGATION OF THE SOLUTION OF SPARSE MATRICES IN CATHENA

T.G. Beuthe and J.B. Hedley

Atomic Energy of Canada Limited
Whiteshell Laboratories
Safety Thermalhydraulics Branch
Pinawa, Manitoba
Canada R0E 1L0

ABSTRACT

CATHENA presently uses the Harwell MA28 routines to solve the sparse matrices generated by the thermalhydraulics section of the code. The objective of this paper is to present a short overview of commonly used sparse matrix solution techniques, and examine the potential benefits of using other solvers in CATHENA. A number of direct and iterative solvers have been tested on a stand-alone basis. Their performance was compared by solving CATHENA-generated non-positive definite, non-symmetric matrices taken from a variety of different simulations. The results of this preliminary investigation tend to indicate that some solvers may have advantages over the MA28 routines which could significantly enhance the performance of CATHENA.

1. INTRODUCTION

The solution of systems of linear equations is one of the most important areas of numerical mathematics. A large number of different descriptions of physical problems can be reduced to a linear system of the form:

$$Ax = b$$

where x represents a vector of variables to be solved for, A represents the matrix of coefficients of the linear system, and b represents a vector of constants.

The matrix A is often sparsely populated (less than 10% of the positions are occupied, often significantly less) and the problem can involve the simultaneous solution of a system involving millions of equations. Problems of such magnitude can seriously challenge the computational capabilities of a given computer. They can only be solved using numerical algorithms that take sparseness and structure into account, and use special storage and programming techniques.

Matrices to be solved may have real or complex elements, may be symmetric or unsymmetric (for real matrices) or Hermitian or non-Hermitian (for complex matrices). They may be positive definite, banded, have a block structure, or be diagonally dominant. Depending on which of these characteristics a given matrix displays, special algorithms have been developed to solve the

system of equations with the required accuracy. These algorithms save storage and computation time. Since research problems are typically working to the limit of available computational resources, exploitation of matrix sparseness can make the difference between solving and not solving a given problem.

2. SOLUTION METHODS

2.1 Direct Methods

Direct methods yield the required solution with fixed number of arithmetic operations. At the heart of every direct method lies the Gaussian elimination process and the related triangular decomposition. Variants of the basic direct method differ primarily in the way the matrix A is stored, the details of the elimination process, the precautions used to minimize rounding errors, and the methods of refining solutions.

Pivoting (exchanging rows and/or columns) often helps to assure the numerical accuracy and stability of direct methods by moving the largest elements into the diagonal. Typically, attempts are made to minimize the fill-in of new non-zero values generated as a result of the solution process. Some routines offer a drop-tolerance to remove insignificant fill-in values and residual refinement to improve the accuracy of the final answer. Direct methods are noted for their relative robustness, ease of use, and ease of acquisition. Good general discussions of direct methods for sparse matrices can be found in the books by Pissanetzky [1], Duff et al. [2], and Zlatev [3].

2.2 Iterative Methods

Iterative methods begin with a starting vector x^0, and compute a sequence of iterands x^m for $m = 1, 2, 3, \ldots$

$$x^0 \mapsto x^1 \mapsto x^2 \mapsto x^3 \mapsto \cdots \mapsto x^{m-1} \mapsto x^m \mapsto x^{m+1} \mapsto \cdots$$

where x^{m+1} is dependent only on x^m, and starting value x^0 is not part of the method.

A preconditioner is often used to help speed up the convergence of the iterations. The term derives its name from the idea that an improvement in the condition number of matrix A helps the iteration proceed. An undesirable side effect of this process lies in the potentially large amount of time that can be spent in the preconditioning stage.

A variety of different methods exist, some of which only work for specialized classes of matrices or problems. Good generalized robust solvers are more difficult to find, and are often only available on a commercial basis. They can be complicated to use, and often include a large number of options which require the use of expert technical support.

Nonetheless, iterative methods can be advantageous for sparse matrices since far fewer calculations are performed per iteration than are made during the solution using direct methods.

Additional advantages exist if a good approximation to x is already available to accelerate the convergence. If the matrix has a block structure, this can also be used to help improve the efficiency of the iterative solver. Unfortunately, if a matrix is not positive definite, convergence is not guaranteed when iterative methods are used. Comprehensive summaries of iterative techniques can be found in the books by Ilin [4] and Hackbusch [5], and the article by Dongarra, and van der Vorst [6].

2.3 Parallel Solvers

Parallel solvers solve different parts of the matrix simultaneously. These methods also often take advantage of available vector processing, and typically avail themselves of methods developed for direct and iterative solvers. Two general approaches are used to solve matrices in parallel:

1. Consider the inherent ability of the detailed coding to be performed in parallel, or by using vector processors. For example, row and column swapping might be done in parallel, or a section of the code might be rewritten to vectorize basic matrix operations.
2. Divide the matrix into sub-groups that can be calculated separately. Some commonly used techniques include partitioning, matrix modification, and tearing. These methods tend to perturb the matrix, but matrix perturbation techniques may also be used to make matrices easier to solve.

Parallel methods represent the forefront of development work in matrix solution. As new machines and hardware become available, more and more parallel methods are being explored. At this point in time, parallel solvers can be as varied as the hardware they are implemented on. Although no books have been written explicitly on parallel matrix solvers, books like those by Schendel [7] and Evans [8] include discussions of parallel methods.

3. THE CATHENA CODE

CATHENA (Canadian Algorithm for THErmalhydraulic Network Analysis) is a one-dimensional, two-fluid thermalhydraulic computer code designed for the analysis of two-phase flow and heat transfer in piping networks. The CATHENA thermalhydraulic code was developed by AECL, Whiteshell Laboratories, primarily for the analysis of postulated accident conditions in CANDU® reactors.

The thermalhydraulic model in CATHENA is a one-dimensional, non-equilibrium two-fluid model consisting of six partial differential equations for mass, momentum and energy conservation – three for each phase. A first-order finite-difference representation is used to solve the differential equations, utilizing a semi-implicit one-step method in which the time step is not limited to the

---

CANDU® is a registered trademark of Atomic Energy of Canada Limited (AECL).

material Courant number [9]. At each time step a sparse matrix is written and is currently solved using the Harwell MA28 sparse matrix solver which was developed in the early 1970s.

When relatively small simulations are performed, and the order of the matrix to be solved n is less than 4000, less than 25% of the total simulation time is usually spent in the sparse solver. In general, the computational effort needed to assemble the CATHENA sparse matrix scales as $O\{n\}$. In contrast, the effort needed to solve the sparse matrix is less than $O\{n^2\}$ but still far from linear with n however, depending on the efficiency of the solver, and the sparsity and structure of the matrix [2]. As a result, the fraction of the total time spent solving the generated sparse matrices increases as the size of the simulation increases. As shown in Figure 1, as the order of the matrix increases to greater than 17,000 more than 90% of the simulation can be spent in solving the sparse matrices, and the efficiency of the sparse solver can become the predominant factor affecting the computational efficiency of CATHENA.

Although the CATHENA sparse matrices are non-symmetric, non-positive definite, and relatively stiff and intractable to solution, the MA28 routines have proven themselves robust and reliable for more than 10 years. In the interim however, a number of potentially more efficient solvers have become available. In this study, 7 such solvers were obtained from public and commercial sources. A variety of different test matrices were generated by CATHENA. The size of the test matrices was varied from small to the largest possible within the present limits of CATHENA, as shown in Table 1. Examples of the structure of the sparse matrices are shown in Figures 2 and 3. These matrices were used to test and compare the performance of the assembled solvers on a stand-alone basis. What follows is a summary of the test results.

4. TEST OF DIRECT SOLVERS

For this test series, small interface codes were assembled using the sparse matrix solvers commercially available in the NAG [10] and IMSL [11] libraries. Scientific Computing Associates Inc. (SCA), a company in New Haven, Connecticut specializing in the production of advanced direct, iterative and parallel sparse matrix solvers was kind enough to make their direct solver SMPAK available for these tests. The publicly available Y12M and NSPIV routines were also tested, and the Harwell MA28 routines were included to serve as a benchmark of the current performance of CATHENA.

A summary of the timing results in hundredths of a second is shown in Table 2 for all direct solvers and all matrices. In general, the NSPIV solver was much slower than MA28, the Y12M, NAG, and IMSL routines performed as well as the MA28 routines, and the SMPAK routines typically outperformed MA28 routines.

The relative performance of each solver with respect to MA28 is shown in Table 3. As shown here, there are some surprising exceptions to the general trends. Although, the NSPIV routines typically do not perform well with respect to the MA28 routines, the CWIT and LASH test matrices were solved more than 4 times faster with the NSPIV than with the MA28 routines. The SMPAK routines were typically twice as fast as the MA28 routines when solving the smaller matrices. For the large LEPBIG matrix, SMPAK was almost 18 times as fast as MA28. On the

other hand, SMPAK was quite slow in solving the LASH matrix which was solved so efficiently by NSPIV. The NAG routines performed almost as well as the MA28 routines in all cases. The Y12M and IMSL routines also typically performed as fast as the MA28 routines for smaller matrices, but like the SMPAK routines, their performance improved dramatically in comparison to the MA28 routines when solving larger matrices.

5. TEST OF ITERATIVE SOLVERS

As mentioned above, CATHENA generates non-symmetric, non-positive definite matrices that are not easily amenable to solution by iterative solvers. Two commercial packages were found with the capability to solve CATHENA matrices: the PCGPAK3 routines offered by the Scientific Computing Associates, and the MATB routines offered by Peter Sutherland at the University of Waterloo. Again, SCA was kind enough to offer offer their suite of iterative solvers for testing, and an older version of the MATB routines were located for testing.

Although both packages offer a variety of options, the tests were performed in both cases using an incomplete LU preconditioner and a GMRES iteration method. This proved to be the most efficient and stable combination. Block solution was not used in either package, the input parameters were set the same for each test, and a vector filled with zeros was used as the initial vector for the iteration. As with the direct solvers, small interface codes were created to access the PCGPAK3 and MATB routines and solve the CATHENA test matrices on a stand-alone basis.

The results for the PCGPAK3 and MATB routines are shown in Table 4. As shown here, the MATB routines proved to be relatively inefficient at solving CATHENA matrices. Discussion with the author revealed that the MATB is probably unsuited to this class of matrices. In fact, it was not possible to solve the U1 and CANDU9 matrices with MATB, most likely due to the presence of zeros located on the diagonal. Although the iteration stage of the MATB package was relatively efficient, most of the solution time was spent in the analysis and preconditioning stages.

The PCGPAK3 package on the other hand proved itself to be relatively efficient in comparison to MA28. This is especially true in light of the fact that the original structure of the matrix provided by CATHENA has been optimized to run most efficiently on the MA28 routines. Thus, it was not possible to take advantage of structure optimization or the use of a good first guess at x^0 which would be available during a CATHENA simulation.

It should also be noted that both the MATB and PCGPAK3 routines make use of the Basic Linear Algebra Subroutines (BLAS). These routines are available as standard high-level language coded routines, as well as machine coded routines which are optimized for a particular machine hardware. A short study indicates that a savings of more than 20% in the run times can be achieved over and above the times shown in Table 4 through the use of specially optimized BLAS routines.

6. CONCLUSIONS

Although the MA28 routines have been available since the early 1970s, they are still competitive with many of the more recently available routines. However, a significant performance enhancement could be obtained in CATHENA through a simple replacement of the sparse matrix solver, especially for large simulations. In these cases it may be possible to provide a more than ten-fold performance enhancement. The next step in this effort will be to implement the most promising solvers directly into CATHENA and make in-situ accuracy and performance tests.

7. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the help and cooperation of Frank Stanchell, Applied Geosciences Branch, AECL/WL, and the financial support of the AECL Centre for Mathematical Sciences in the early part of this study.

REFERENCES

1. S. Pissanetzky, Sparse Matrix Technology, Academic Press, London, 1984.
2. I.S. Duff, A.M. Erisman, and J.K. Reid, Direct Methods for Sparse Matrices, Oxford University Press, New York, 1986.
3. Z. Zlatev, Computational Methods for General Sparse Matrices, Kluwer Academic, Boston, 1991.
4. V.P. Ilin, Iterative Incomplete Factorization Methods, World Scientific Pub. Co., River Edge, NJ, 1992.
5. W. Hackbusch, Iterative Solution of Large Sparse Systems of Equations, Springer-Verlag, New York, 1994.
6. J.J. Dongarra, and H.A. van der Vorst, Supercomputer, v. 9, n. 5(1992)17–30.
7. U. Schendel, Sparse Matrices: Numerical Aspects with Applications for Scientists and Engineers, Halsted Press, John Wiley & Sons, New York, 1989.
8. D.J. Evans, Sparsity and Its Applications, University Press, New York, 1985.
9. B.N. Hanna, "CATHENA: A Thermalhydraulic Code for CANDU Analysis", Nuclear Engineering and Design (accepted for publication), 1997.
10. NAG Fortran Library Manual, NAG Inc., Downers Grove, IL, USA, 1993.
11. IMSL User's Manual, FORTRAN Subroutines for Mathematical Applications, IMSL Inc., Houston TX, USA, 1990.

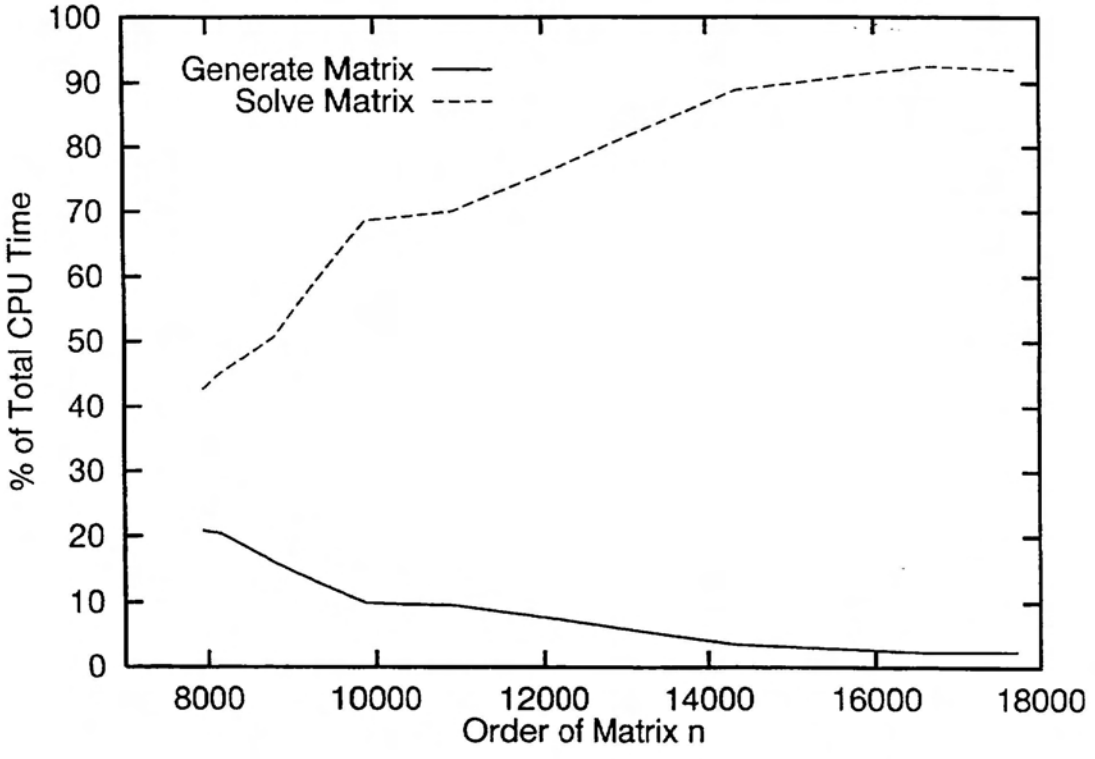

FIGURE 1: Relative amount of time needed to generate and solve CATHENA sparse matrices as a function of the order of the matrix n.

TABLE 1: Summary of matrices.

Matrix	Order	#Non-Zero	% Sparsity
Edwards	64	163	4.0
7-SISTERS	754	2481	0.44
CWIT	1534	4063	0.17
RD14	1654	5701	0.21
U1	2272	7750	0.15
LASH	2292	6248	0.12
RD12	2390	8199	0.14
RD14M	3168	10741	0.11
BTF	3373	10762	0.095
IRF	3846	12641	0.085
CANDU 9	4241	17503	0.097
LEPSMALL	7941	27456	0.044
LEPBIG	17733	80836	0.026

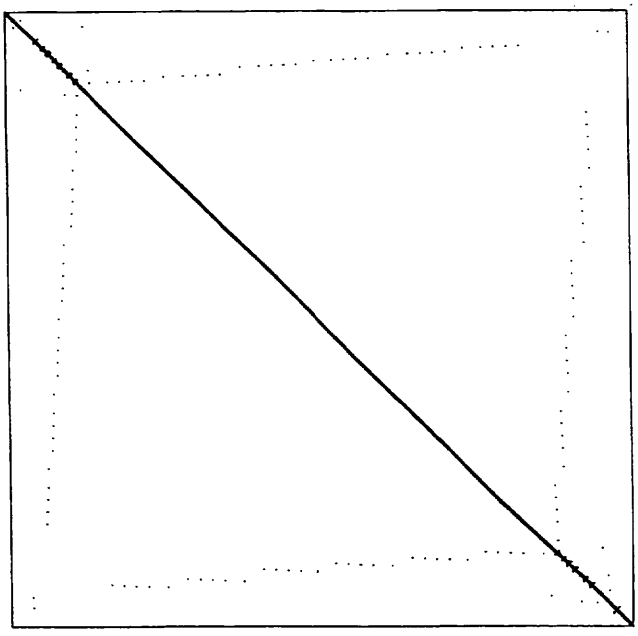

FIGURE 2: Structure of the LASH matrix.

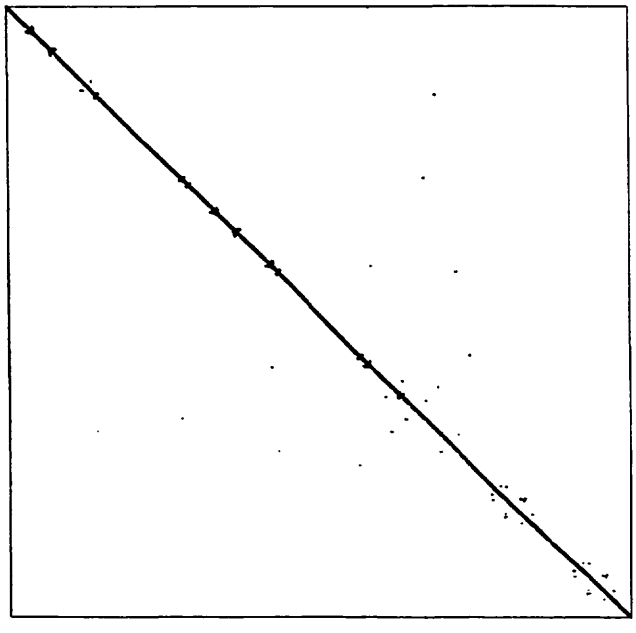

FIGURE 3: Structure of the RD12 matrix.

TABLE 2: Solution times for direct solvers (absolute, hundredths of seconds)

Test	MA28	SMPAK	Y12M	NAG	NSPIV	IMSL
Edwards	2	1	2	2	1	1
7-SISTERS	29	7	27	27	50	29
CWIT	45	19	48	51	11	45
RD14	91	38	79	106	2168	91
U1	195	180	103	229	5391	170
LASH	86	491	79	99	18	86
RD12	132	57	117	165	6176	135
RD14M	259	84	162	341	5116	228
BTF	132	117	138	160	117	142
IRF	410	294	192	452	1438	315
CANDU 9	1111	465	282	1113	88076	416
LEPSMALL	882	543	381	932	127248	549
LEPBIG	26577	1503	1245	28043	–	3012

TABLE 3: Relative performance of direct solvers (MA28/x)

Test	MA28	SMPAK	Y12M	NAG	NSPIV	IMSL
Edwards	1	2.0	1.0	1.0	2.0	2.0
7-SISTERS	1	4.1	1.1	1.1	0.58	1.0
CWIT	1	2.4	0.94	0.88	4.1	1.0
RD14	1	2.4	1.2	0.86	0.042	1.0
U1	1	1.1	1.9	0.85	0.036	1.1
LASH	1	0.18	1.1	0.87	4.8	1.0
RD12	1	2.3	1.1	0.80	0.042	1.0
RD14M	1	3.1	1.6	0.76	0.051	1.1
BTF	1	1.1	1.0	0.83	1.1	0.93
IRF	1	1.4	2.1	0.91	0.29	1.3
CANDU 9	1	2.4	3.9	1.0	0.013	2.7
LEPSMALL	1	1.6	2.3	0.95	0.0069	1.6
LEPBIG	1	17.7	21.3	0.95	–	8.8

TABLE 4: Solution times for iterative solvers (hundredths of seconds), and performance of routines relative to MA28. [n]=# of iterations.

Test	MA28	PCGPAK3	MATB	MA28/PCGPAK3	MA28/MATB
Edwards	2	5 [3]	320 [3]	0.40	0.0063
7-SISTERS	29	50 [5]	320 [1]	0.58	0.091
CWIT	45	71 [3]	970 [85]	0.63	0.046
RD14	91	122 [5]	370 [1]	0.75	0.25
U1	195	277 [5]	–	0.70	–
LASH	86	507 [4]	520 [13]	0.17	0.17
RD12	132	204 [5]	400 [1]	0.65	0.33
RD14M	259	316 [5]	450 [1]	0.82	0.58
BTF	132	279 [3]	630 [9]	0.47	0.21
IRF	410	773 [9]	520 [2]	0.53	0.79
CANDU 9	1111	701 [8]	–	1.6	–
LEPSMALL	882	1102 [9]	1050 [5]	0.80	0.84
LEPBIG	26577	2414 [8]	2650 [5]	11.0	10.0