20th Annual Conference of the Canadian Nuclear Society
Montreal, Quebec, Canada / May 30 - June 2, 1999

THE SOLUTION OF SPARSE MATRICES IN CATHENA

T.G. Beuthe and J.B. Hedley

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

ABSTRACT

CATHENA presently uses the Harwell MA28 routines to solve the sparse matrices generated by the thermalhydraulics numerical method. The objective of this paper is to present an overview of commonly used sparse matrix solution techniques, and to examine the potential benefits of using other solvers in CATHENA. Previous out-board tests have shown that the SMPAK, Y12M, and IMSL direct solvers and the PCGPAK3 iterative solver may be competitive with the MA28 solver. In the present investigation the performance of these solvers was tested in-situ in CATHENA using a wide variety of different simulations. The results indicate the SMPAK and Y12M direct solvers show the best performance and can increase the overall simulation speed by up to a factor of 8 for the largest CATHENA simulations tested.

1. INTRODUCTION

CATHENA (Canadian Algorithm for THErmalhydraulic Network Analysis) is a computer program 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 employed in CATHENA uses 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 material Courant number [1]. At each time step the linearization of the differential conservation equations results in a sparse matrix which is written and solved. Currently the solution is provided using the Harwell MA28 sparse matrix solver which was developed in the early 1970s.

When relatively small network simulations are performed, for example when 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 linearly with the order of the matrix as n. In contrast, the computational effort needed to solve a sparse matrix scales less than n^2 but is still far from linear with n, depending on the efficiency of the solver, the sparsity and the 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. Earlier studies have shown that as the order of the CATHENA matrix increases to greater than 17,000 more than 90% of the simulation can be spent in solving the sparse matrices. In these cases, the efficiency of the sparse solver can become a dominant factor in the computational efficiency of CATHENA [3].

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

Although the CATHENA sparse matrices are non-symmetric, non-positive definite, and relatively stiff, the MA28 routines have proven themselves robust and reliable for more than 10 years. In the interim a number of potentially more efficient solvers have become available. A preliminary study where CATHENA generated sparse matrices were solved on a stand-alone basis, using 6 direct and 2 iterative sparse matrix solvers, showed that it may be possible to achieve significant savings in solution time through the use of alternative matrix solvers [3].

After presenting a review of the available sparse matrix solution techniques, the present study examines five (5) of the most promising matrix solvers (4 direct and 1 iterative) investigated in the preliminary study. These solvers were directly implemented in a test version of CATHENA MOD-3.5c/Rev 0 and were used to solve a wide range of thermalhydraulic simulations, from very small and simple to extremely large and complex. The solvers under examination were tested for both accuracy and speed. The results of this investigation indicate that two of the direct solver solvers are much faster than the MA28 routines and significantly enhance the overall performance of CATHENA for large simulations.

2. REVIEW OF SPARSE MATRIX SOLUTION TECHNIQUES

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:

$$Ax = b \tag{1}$$

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 any given machine. 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 to minimize storage and computation time.

2.1 Solution Methods

Solution methods for sparse matrices can be classified into one of three general categories:

1. **Direct Methods** that yield the required solution with a fixed number of arithmetic operations.
2. **Iterative Methods** that 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 only dependent on x^m, and starting value x^0 is not part of the method.

3. **Parallel Solvers** that solve 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.

2.2 Direct Methods

At the heart of every direct method lies the Gaussian elimination process and the related triangular decomposition. To solve equation 1, a decomposition of matrix A into lower L and upper U triangular matrices is performed:

$$A = LU \tag{2}$$

and a forward and back-substitution is performed to find the solution vector x:

$$LUx = b \;\Leftrightarrow\; Ly = b \;\; Ux = y \tag{3}$$

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.

Special direct methods exist for symmetric or positive definite matrices that need only about half the number of storage cells. A special direct method known as the **frontal technique** also exists. Although originally developed for finite element analysis, it is not restricted to that application [4, 5].

The numerical accuracy and stability of direct methods is normally assured by moving the largest elements into the diagonal through row and column exchanges, an operation called **pivoting**. **Partial pivoting** involves exchanging only rows and **full pivoting** involves exchanging both rows and columns. The usefulness of pivoting is not always guaranteed due to the time and effort it takes to perform the pivoting operations [6].

In the process of solving the matrix A, new non-zero elements, known as fill-ins, will be created. Direct solvers attempt to minimize the number of fill-ins wherever possible. Some of these new elements will be physically significant, whereas others could theoretically be dismissed as numerical roundoff. Some routines make provisions for dropping these insignificant values through a drop-tolerance parameter. Depending on the drop-tolerance used, it may be necessary to improve the accuracy of the final answer in a process known as residual refinement [2, 7].

Direct solution methods have received a significant amount of attention in the literature. Recent discussion on the use of direct methods in fluid mechanics problems can be found in articles by Onyejekwe et al [8] on fluid flow in pipe networks, and Habashi et al [9] on the use of direct methods well suited for use on supercomputers. The use of direct solvers in finite element problems is discussed by Leimbach and Zeller [10] (for the nuclear industry) and Peters [11].

If matrix A is not well conditioned, direct methods can sometimes succeed where iterative methods can fail. Young et al [12] discuss a case in which direct solvers were chosen over iterative solvers for intractable problems in the aerospace industry.

Due to their relative robustness, good track record, and ability to solve a matrix in a finite number of steps, direct solvers such as MA28 and Y12M [13, 14] have been in use for some time now, and are looked upon almost as an industry standard. As such, these routines are often used as baselines for comparisons between other routines. For example, Duff and Nowak compare the performance of NSPFAC and MA28 in the LARKIN program [15]. Good general discussions of direct methods for sparse matrices can be found in the books by Pissanetzky [4], Duff et al [2], and Zlatev [7]. The book by Duff represents a more general

introduction to direct methods. The book by Zlatev is a more extensive publication discussing a wide range of subject areas.

2.3 Iterative Methods

Some of the more commonly known iterative method include

- Jacobi, Gauss-Seidel, and Successive Overrelaxed (SOR) methods. These are sometimes referred to as classical methods.
- Conjugate-Gradient Methods. Although they are very popular, these algorithms unfortunately require positive definiteness in matrix A.
- Multi-Grid Methods. Unlike the previous routines which have at best a linear convergence, multi-grid methods have a convergence which is independent of step-size.
- Domain Decomposition.

In general, the $\tilde{x}_{m+1}$ vector in iterative methods is only dependent on the x vector from the preceeding step $\tilde{x}_m$, as well as matrix A and constant vector b:

$$\tilde{x}_{m+1} = \Phi(\tilde{x}_m) \quad \text{where} \quad \Phi = f(A, b) \tag{4}$$

Semi-iterative methods also exist. In this case, $\tilde{x}_{m+1}$ is calculated using more than just $\tilde{x}_m$:

$$\tilde{x}_{m+1} = \Phi(\tilde{x}_m, \tilde{x}_{m-1}, \tilde{x}_{m-2} \ldots) \tag{5}$$

An example of such a method is the Alternating-Direction Implicit or ADI 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 [16]. Nonetheless, iterative methods can be advantageous for sparse matrices since far fewer calculations are performed per iteration than are made during the solution when using direct methods. Additional advantages exist if a good approximation to x is already available to accelerate the convergence. Unfortunately, if a matrix is not positive definite, convergence is not guaranteed.

Comprehensive summaries of iterative techniques can be found in the books by Ilin [17] and Hackbusch [18]. The book by Hackbusch in particular stands out as a good and very up-to-date overview of iterative methods. A comparative summary of various iterative techniques can be found in the article by Dongarra and van der Vorst [19].

2.4 Parallel 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 separately calculated. 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 better condition the matrix.

Parallel methods represent the forefront of development work in matrix solution. As new machines and hardware become available, new parallel methods are developed. Further details can be found in references [20–24]. The present investigation will only consider the use of direct or iterative techniques.

3. SOLVER TESTING

3.1 Solver Implementation and Choice of Test Cases

In an earlier investigation [3], a broad pallet of sparse matrix solvers was tested on a small number of CATHENA matrices. These tests were performed on an outboard basis and were benchmarked against the MA28 solver. The results indicated the direct solvers in the Y12M routines by Zlatev et al [13], the commercial IMSL routines, and the specialized SMPAK routines developed by Scientific Computing Associates (SCA) deserved further investigation. Of the iterative solvers tested, only the set of results from the PCGPAK3 routines offered by SCA warranted further investigation.

For the present investigation, these routines were integrated into a test version of CATHENA and used to run a wide range of test cases. As summarized in Table 1, the test cases ranged from the smallest cases such as TEST1 (16 equations and 34 non-zero terms) to some of the largest and most complex simulations being performed with CATHENA. The standard CATHENA acceptance test suite was used to ensure the solvers had been correctly integrated into the test version of CATHENA and were producing results consistent with MA28 results.

Wherever possible, the sparse solvers were implemented using the standard recommended preset values. The IMSL sparse solver was implemented in such a way that it could be utilized in row pivot, column pivot as well as row plus column pivot mode.

The iterative solver, PCGPAK3, which in fact represents a suite of iterative solution methods, was implemented using an incomplete LU preconditioner and a Generalized Minimal RESidual (GMRES) iteration method. This proved to be the most efficient and stable combination. Block solution was not used. The initial guess on the first step was given by the intial conditions provided by CATHENA. Initial guesses for subsequent steps were provided by the solution from the previous time step, thus fully utilizing this accelerative feature of iterative solvers.

Care was taken to ensure the test cases chosen in Table 1 represent a good cross section of the types of sparse matrices generated by CATHENA. As shown in Figures 1–4, the test cases chosen show a wide range of structure. The structure of the CATHENA-generated sparse matrices is largely dependent on the manner in which the network connections are assembled in the CATHENA input file by the user and thus tend to reflect both the structure of the physical facility and the approach used to idealize it. For example, the TEST20 matrix shown in Figure 4 represents an idealization of the RD-14M facility [25]. The square symmetric off-diagonal non-zero entries (as represented by the solid points in the figure) at the top left and the middle reflect the presence of the two sets of parallel heated sections in the facility. The clusters of off-diagonal non-zero entries at the bottom right represent the steam generators and the ECI system.

3.2 Test Results

The results of the tests are shown in Tables 2 and 3. Table 2 summarizes the average matrix solution time (in seconds) per CATHENA timestep, and Table 3 shows the performance of the solvers relative to the performance of MA28. As shown here, the SMPAK direct solver has the best overall performance relative to MA28 for both small and large cases. With the exception of the TEST17 case, the SMPAK routine is significantly faster than MA28. In one case (TEST25, one of the largest test cases) the SMPAK routines take only 2.3% of the time it takes MA28 to solve the CATHENA sparse matrices.

The next best performance was provided by the Y12M solver. Although it does not do as well with the small cases, it meets, and in one case (TEST24) even slightly exceeds the performance of SMPAK for the larger cases.

The IMSL sparse solver routine was tested in column, row, and row plus column pivot mode. For small cases, the IMSL routine performance was close to that of MA28, but in many cases it was slower. In general the choice of pivoting does not have a large effect on the performance of the routine when solving CATHENA matrices. Like the Y12M routine, the IMSL routine provided the best performance relative to the MA28 routine when solving large matrices. However, the overall performance was nowhere near that of the SMPAK and Y12M routines for CATHENA matrices.

On average, the iterative PCGPAK3 routine was about as fast as the MA28 solver. In some cases it was significantly slower (by a factor of up to 2.5). The PCGPAK3 routines were quite competitive with the MA28 solver for the larger matrices, but the performance for these cases was 2–4 times slower than the SMPAK or Y12M routines.

It should be noted that the 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. Previous studies indicate savings of more than 20% in the run times can be achieved through the use of specially optimized BLAS routines [3]. This does not provide a large enough saving to make the PCGPAK3 routines competitive with direct solver routines like Y12M or SMPAK.

4. SUMMARY AND CONCLUSIONS

The MA28 solver is presently used as the standard sparse matrix solver in CATHENA. This solver is still competitive with newer routines when solving small to medium sized CATHENA matrices (order < 4000).

Recently, users have begun to create larger simulations. These typically model an entire CANDU reactor, including subsystems, with an increasing degree of detail. In these large simulations, a significant performance enhancement could be obtained through a simple replacement of the sparse matrix solver. The overall run times of the cases could be significantly reduced. For example, the TEST24 case spends more than 70% of its time in the MA28 sparse solver. A switch to the SMPAK solver would increase the overall performance of the code by a factor of more than 2.5. The TEST25 case, which spends 90% of its time in the MA28 solver, would run more than 8 times faster if the SMPAK solver was used instead.

The present study indicates the two routines which provide the best performance in comparison to the MA28 routine for solving CATHENA generated sparse matrices are the direct sparse matrix solvers: Y12M and SMPAK. Implementation of these routines is recommended as alternative sparse matrix solvers for future CATHENA versions.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the help and cooperation of D. Baxter at SCA for his help with the SMPAK and PCGPAK3 routines.

REFERENCES

1. B.N. Hanna, “CATHENA: A thermalhydraulic code for CANDU analysis”, Nuclear Engineering and Design, 180 (1998) 113-131.
2. I.S. Duff, A.M. Erisman, and J.K. Reid, Direct Methods for Sparse Matrices, Oxford University Press, New York, 1986.
3. T.G. Beuthe and J.B Hedley, “Preliminary Investigation of the Solution of Sparse Matrices in CATHENA”, Proceedings of the 20th Nuclear Simulation Symposium, Niagara-on-the-Lake, Ontario, 1997 September 7–9.
4. S. Pissanetzky, Sparse Matrix Technology, Academic Press, London, 1984.
5. P. Hood, Int. J. Num. Meth. Eng., 10(1976)379–399.
6. S. Thompson, The Effect of Partial Pivoting in Sparse Ordinary Differential Equation Solvers, Comp. & Math. with Appl. Ser. A, v. 12A, n. 12(Dec. 1986)1183–91.
7. Z. Zlatev, Computational Methods for General Sparse Matrices, Kluwer Academic, Boston, 1991.
8. O.O. Onyejekwe, A. Okoromadu, and V. Onyema, Advances in Engineering Software, 17(1993)189–194.
9. W.G. Habashi, V.-N Nguyen, M.V. Bhat, Comp. Meth. Appl. Mech. & Eng. 87(1991)253–265.
10. K.R. Leimbach, C. Zeller, Advantages of Using a Node-Oriented Sparse-Matrix Solver in Nuclear Plant Dynamic Analyses, Nucl. Eng. & Des. v. 70, n. 1(1982)85-100.
11. F.J. Peters, Sparse Matrices and Substructures with a Novel Implementation of Finite Element Algorithms, Amsterdam, Mathematisch Centrum, 1980.
12. D.P. Young, R.G. Melvin, F.T. Johnson, J.E. Bussoletti, L.B. Wigton, and S.S. Samant, SIAM J. Sci. Stat. Comput., v. 10 n. 6(Nov. 1989)1186–1199.
13. Z. Zlatev, J. Wasniewski, K. Schaumburg, Y12M solution of large and sparse systems of linear algebraic equations : documentation of subroutines, Springer-Verlag, New York, 1981.
14. O. Sterby, and Z. Zlatev, Direct methods for sparse matrices, Springer-Verlag, New York, 1983.
15. I.S. Duff, and U. Nowak, IMA J. Num. Anal. 7(1987)391–405.
16. G. Brussino, V. Sonnad, Int. J. Num. Meth. Eng. 28(1989)801–815.
17. V.P. Ilin, Iterative Incomplete Factorization Methods, World Scientific Pub. Co., River Edge, NJ, 1992.
18. W. Hackbusch, Iterative Solution of Large Sparse Systems of Equations, Springer-Verlag, New York, 1994.
19. J.J. Dongarra, and H.A. van der Vorst, Supercomputer, v. 9, n. 5(1992)17–30.
20. J.N. Shadid, and R.S. Tuminaro, Concurrency: Practice and Experience, v. 4, n. 6 (1992) 481–497.
21. U. Schendel, Sparse Matrices: Numerical Aspects with Applications for Scientists and Engineers, Halsted Press, John Wiley & Sons, New York, 1989.
22. D.J. Evans, Sparsity and Its Applications, University Press, New York, 1985.
23. X. Xu, N. Qin, and B.E. Richards, Int. J. Num. Meth. Fluids 15(1992)613–623.
24. J. Larsen, and T. Christensen, A Parallel Sparse Matrix Solver, Proc. of the 1st & 2nd Nordic Transputer Seminars, (1991)64–75.
25. P.J. Ingham, J.C. Luxat, A.J. Melnyk, T.V. Sanderson, Natural Circulation Experiments in an Integral CANDU Test Facility, IAEA Technical Committee Meeting on Experimental Tests and Qualification of Analytical Methods to Address Thermohydraulic Phenomena in Advanced Water Cooled Reactors, Paul Scherrer Institute, Villigen, Switzerland, 1998 September 14–17.

TABLE 1: Summary of test matrices.

Test	Order	#Non-Zero	% Sparsity
TEST1	16	34	13.28125
TEST2	39	205	13.47798
TEST3	94	586	6.63196
TEST4	130	447	2.64497
TEST5	142	805	3.99226
TEST6	142	918	4.55267
TEST7	194	680	1.80678
TEST8	202	1036	2.53897
TEST9	336	1191	1.05495
TEST10	368	1994	1.47241
TEST11	527	1842	0.66324
TEST12	754	2730	0.48020
TEST13	842	3602	0.50807
TEST14	884	3598	0.46042
TEST15	1368	5116	0.27337
TEST16	1654	6322	0.23109
TEST17	2292	10530	0.20045
TEST18	2390	9492	0.16617
TEST19	3008	18714	0.20683
TEST20	3168	15926	0.15869
TEST21	3846	13886	0.09388
TEST22	8009	32453	0.05059
TEST23	13391	45158	0.02518
TEST24	19470	88935	0.02346
TEST25	17733	91929	0.02923

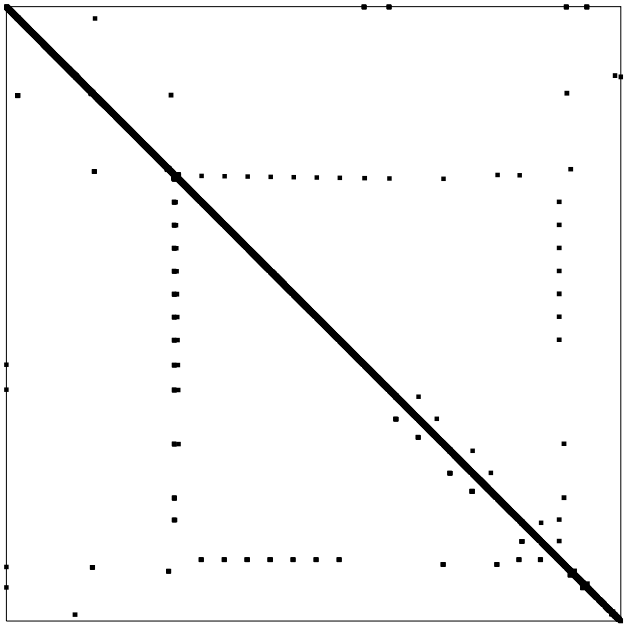

FIGURE 1: Structure of the TEST21 matrix.

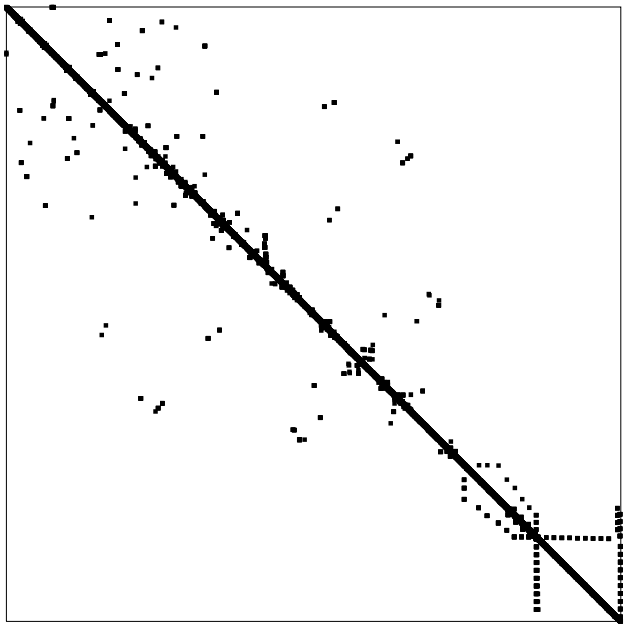

FIGURE 2: Structure of the TEST22 matrix.

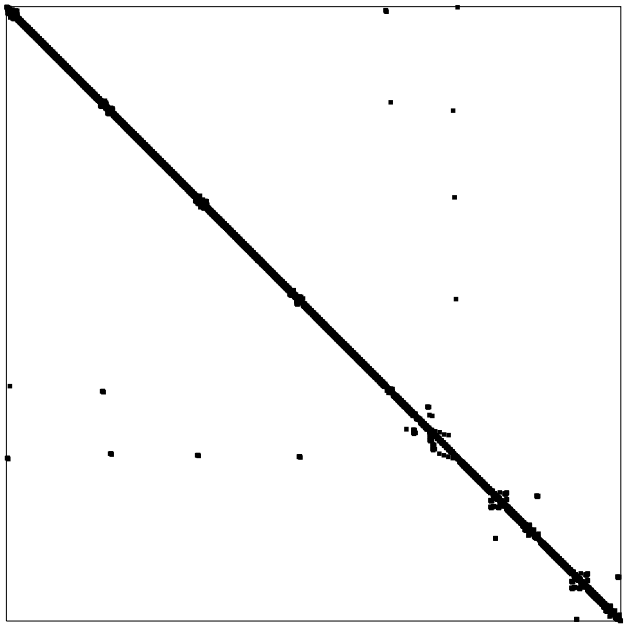

FIGURE 3: Structure of the TEST16 matrix.

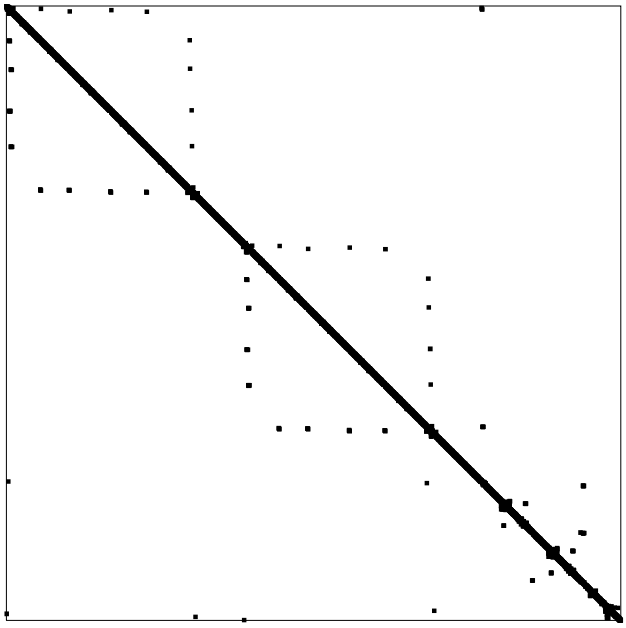

FIGURE 4: Structure of the TEST20 matrix.

TABLE 2: Average matrix solution times per CATHENA timestep in seconds.

Test	MA28	SMPAK	Y12M	IMSL^{C}	IMSL^{R}	IMSL^{R+C}	PCGPAK3
TEST1	0.00164	0.00024	0.00136	0.00219	0.00239	0.00243	0.00216
TEST2	0.00625	0.00109	0.00603	0.00871	0.00832	0.00927	0.00601
TEST3	0.02034	0.00328	0.01445	0.02000	0.02020	0.02184	0.01328
TEST4	0.01048	0.00247	0.01034	0.01504	0.01556	0.01857	0.01314
TEST5	0.04117	0.00636	0.02586	0.03358	0.03268	0.03430	0.01799
TEST6	0.04488	0.00717	0.03047	0.05041	0.04429	0.04535	0.01911
TEST7	0.01578	0.00418	0.01549	0.02265	0.02194	0.02472	0.01907
TEST8	0.02802	0.00218	0.02273	0.03194	0.03514	0.03410	0.01865
TEST9	0.10086	0.02786	0.05912	0.08664	0.08045	0.08742	0.06338
TEST10	0.12494	0.01223	0.07180	0.08900	0.07415	0.07822	0.04422
TEST11	0.05196	0.01924	0.05009	0.06896	0.07863	0.07169	0.08833
TEST12	0.07402	0.01482	0.06661	0.09850	0.09681	0.09796	0.07532
TEST13	0.19925	0.03955	0.15844	0.18965	0.18889	0.18600	0.13238
TEST14	0.33869	0.04363	0.16969	0.22826	0.25000	0.25275	0.14462
TEST15	0.43649	0.12263	0.23328	0.29585	0.33152	0.31634	0.83258
TEST16	0.30557	0.09452	0.24626	0.30990	0.36084	0.34467	0.76422
TEST17	2.06144	6.34248	0.47990	0.72636	0.61338	0.68373	5.29129
TEST18	0.60114	0.14069	0.39721	0.52149	0.58179	0.56019	0.69512
TEST19	6.08560	0.32965	1.02191	1.31847	1.41485	1.33607	1.79781
TEST20	1.91507	0.30064	0.63099	0.94547	0.96651	0.86850	1.40442
TEST21	1.93109	0.66865	0.71353	0.95199	1.12711	0.98651	1.84000
TEST22	3.67701	2.08273	1.48422	2.06059	2.26208	2.11019	2.52024
TEST23	8.25507	2.17492	2.40791	3.49099	3.76920	3.42716	8.19024
TEST24	67.38019	8.89358	4.43000	7.55024	7.73181	6.84090	17.19706
TEST25	159.01497	3.64616	6.93148	7.64363	8.69818	8.84090	11.69482

C = columnwise pivot
R = rowwise pivot
R+C = row and columnwise pivot

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

Test	SMPAK	Y12M	IMSL^{C}	IMSL^{R}	IMSL^{R+C}	PCGPAK3
TEST1	0.150	0.830	1.334	1.459	1.479	1.317
TEST2	0.175	0.964	1.393	1.330	1.482	0.962
TEST3	0.162	0.710	0.983	0.993	1.074	0.653
TEST4	0.236	0.987	1.435	1.484	1.772	1.253
TEST5	0.154	0.628	0.816	0.794	0.833	0.437
TEST6	0.160	0.679	1.123	0.987	1.010	0.426
TEST7	0.265	0.982	1.435	1.391	1.567	1.209
TEST8	0.078	0.811	1.140	1.254	1.217	0.666
TEST9	0.276	0.586	0.859	0.798	0.867	0.628
TEST10	0.098	0.575	0.712	0.593	0.626	0.354
TEST11	0.370	0.964	1.327	1.513	1.380	1.700
TEST12	0.200	0.900	1.331	1.308	1.323	1.017
TEST13	0.199	0.795	0.952	0.948	0.934	0.664
TEST14	0.129	0.501	0.674	0.738	0.746	0.427
TEST15	0.281	0.534	0.678	0.760	0.725	1.907
TEST16	0.309	0.806	1.014	1.181	1.128	2.501
TEST17	3.077	0.233	0.352	0.298	0.332	2.567
TEST18	0.234	0.661	0.867	0.968	0.932	1.156
TEST19	0.054	0.168	0.217	0.232	0.220	0.295
TEST20	0.157	0.329	0.494	0.505	0.454	0.733
TEST21	0.346	0.369	0.493	0.584	0.511	0.953
TEST22	0.566	0.404	0.560	0.615	0.574	0.685
TEST23	0.263	0.292	0.423	0.457	0.415	0.992
TEST24	0.132	0.066	0.112	0.115	0.102	0.255
TEST25	0.023	0.044	0.048	0.055	0.056	0.074

C = columnwise pivot
R = rowwise pivot
R+C = row and columnwise pivot