An Evaluation of Sparse Direct Symmetric Solvers: An Introduction and Preliminary Findings

Jennifer A. Scott[1], Yifan Hu[2], and Nicholas I.M. Gould[1]

[1] Computational Science and Engineering Department
Rutherford Appleton Laboratory
Chilton, Oxfordshire, OX11 0QX, England, UK
{n.i.m.gould,j.a.scott}@rl.ac.uk
[2] Wolfram Research, Inc., 100 Trade Center Drive
Champaign, IL61820, USA
yifanhu@wolfram.com

Abstract. In recent years a number of solvers for the direct solution of large sparse, symmetric linear systems of equations have been developed. These include solvers that are designed for the solution of positive-definite systems as well as those that are principally intended for solving indefinite problems. The available choice can make it difficult for users to know which solver is the most appropriate for their applications. In this study, we use performance profiles as a tool for evaluating and comparing the performance of serial sparse direct solvers on an extensive set of symmetric test problems taken from a range of practical applications.

1 Introduction

Solving linear systems of equations lies at the heart of numerous problems in computational science and engineering. In many cases, particularly when discretizing continuous problems, the system is large and the associated matrix A is sparse. Furthermore, for many applications, the matrix is symmetric; sometimes, such as in finite-element applications, A is positive definite, while in other cases, including constrained optimization and problems involving conservation laws, it is indefinite.

A direct method for solving a sparse linear system $Ax = b$ involves the explicit factorization of the system matrix A (or, more usually, a permutation of A) into the product of lower and upper triangular matrices L and U. In the symmetric case, for positive definite problems $U = L^T$ (Cholesky factorization) or, more generally, $U = DL^T$, where D is a block diagonal matrix with 1×1 and 2×2 blocks. Forward elimination followed by backward substitution completes the solution process for each given right-hand side b. Direct methods are important because of their generality and robustness. Indeed, for the 'tough' linear systems arising from some applications, they are currently the only feasible solution methods. In many other cases, direct methods are often the method of choice because the difficulties involved in finding and computing a good preconditioner for an iterative method can outweigh the cost of using a direct method. Furthermore, direct methods provide an effective means of solving multiple

J. Dongarra, K. Madsen, and J. Waśniewski (Eds.): PARA 2004, LNCS 3732, pp. 818–827, 2006.

systems with the same A but different right-hand sides b because the factorization needs only to be performed once.

Since the early 1990s, many new algorithms and a number of new software packages for solving sparse symmetric systems have been developed. Because a potential user may be bewildered by such choice, our intention to compare the serial solvers (including serial versions of parallel solvers) on a significant set of large test examples from many different application areas. This study is an extension of a recent comparison by Gould and Scott [3] of sparse symmetric direct solvers in the mathematical software library HSL [7]. This earlier study concluded that the best general-purpose HSL package for solving sparse symmetric systems is currently `MA57`. Thus the only HSL direct solver included here is `MA57`, but for some classes of problems, other HSL codes may be more appropriate. For full details and results for the HSL symmetric solvers are given in [4].

The sparse solvers used in this study are listed in Table 1. The codes are discussed in detail in the forthcoming report [5]. Some of the solvers are freely available to academics while to use others it is necessary to purchase a licence. This information is provided in Table 2. For each code a webpage address is also given (or, if no webpage is currently available, an email contact is provided). Note that for non academic users, the conditions for obtaining and using a solver varies between the different packages.

Table 1. Solvers used in our numerical experiments. A '&' indicates both languages are used in the source code; 'F77/F90' indicates there is a F77 version and a F90 version

Code	Date/version	Language	Authors
`BCSLIB-EXT`	11.2001, v4.1	F77	The Boeing Company
`MA57`	01.2002, v1.0.0	F77/F90	I.S. Duff, HSL
`MUMPS`	11.2003, v4.3.2	F90	P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, and J. Koster
`Oblio`	12.2003, v0.7	C++	F. Dobrian and A. Pothen
`PARDISO`	02.2004	F77 & C	O. Schenk and K. Gärtner
`SPOOLES`	1999, v2.2	C	C. Ashcraft and R. Grimes
`SPRSBLKLLT`	1997, v0.5	F77	E.G. Ng and B.W. Peyton
`TAUCS`	08.2003, v2.2	C	S. Toledo
`UMFPACK`	04.2003, v4.1	C	T. Davis
`WSMP`	2003, v1.9.8	F90 & C	A. Gupta and M. Joshi, IBM

2 Test Environment

Our aim is to test the solvers on a wide range of problems from as many different application areas as possible. In collecting test data we imposed only two conditions:

- The matrix must be square and of order greater than $10,000$.
- The data must be available to other users.

Table 2. Academic availability of and contact details for the solvers used in our numerical experiments. * denotes source code is provided in the distribution

Code	Free to academics	Webpage / email contact
BCSLIB-EXT	×	www.boeing.com/phantom/BCSLIB-EXT/index.html
MA57*	×	www.cse.clrc.ac.uk/nag/hsl
MUMPS*	√	www.enseeiht.fr/lima/apo/MUMPS/
Oblio*	√	dobrian@cs.odu.edu or pothen@cs.odu.edu
PARDISO	√	www.computational.unibas.ch/cs/scicomp/software/pardiso
SPOOLES*	√	www.netlib.org/linalg/spooles/spooles.2.2.html
SPRSBLKLLT*	√	EGNg@lbl.gov
TAUCS*	√	www.cs.tau.ac.il/∼stoledo/taucs/
UMFPACK*	√	www.cise.ufl.edu/research/sparse/umfpack/
WSMP	√	www-users.cs.umn.edu/∼agupta/wsmp.html

The first condition was imposed because our interest is in large problems. The second condition was to ensure that our tests could be repeated by other users and, furthermore, it enables other software developers to test their codes on the same set of examples and thus to make comparisons with other solvers. Our test set comprises 88 positive-definite problems and 61 numerically indefinite problems. Numerical experimentation found 5 of the indefinite problems to be structurally singular and a number of others are highly-ill-conditioned. Any matrix for which we only have the sparsity pattern available is included in the positive-definite set and appropriate numerical values generated. Application areas represented by our test set include linear programming, structural engineering, computational fluid dynamics, acoustics, and financial modelling. A full list of the test problems together with a brief description of each is given in [6]. The problems are all available from

ftp://ftp.numerical.rl.ac.uk/pub/matrices/symmetric

In this study, performance profiles are used as a means to evaluate and compare the performance of the solvers on our set $\mathcal{T}$ of test problems. Let $\mathcal{S}$ represent the set of solvers that we wish to compare. Suppose that a given solver $i \in \mathcal{S}$ reports a statistic $s_{ij} \geq 0$ when run on example j from the test set $\mathcal{T}$, and that the smaller this statistic the better the solver is considered to be. For example, s_{ij} might be the CPU time required to solve problem j using solver i. For all problems $j \in \mathcal{T}$, we want to compare the performance of solver i with the performance of the best solver in the set $\mathcal{S}$.

For $j \in \mathcal{T}$, let $\hat{s}_j = \min\{s_{ij}; i \in \mathcal{S}\}$. Then for $\alpha \geq 1$ and each $i \in \mathcal{S}$ we define

$$k(s_{ij}, \hat{s}_j, \alpha) = \begin{cases} 1 \text{ if } \ s_{ij} \leq \alpha \hat{s}_j \\ 0 \text{ otherwise.} \end{cases}$$

The *performance profile* (see [1]) of solver i is then given by the function

$$p_i(\alpha) = \frac{\sum_{j \in \mathcal{T}} k(s_{ij}, \hat{s}_j, \alpha)}{|\mathcal{T}|}, \quad \alpha \geq 1.$$

Thus $p_i(1)$ gives the fraction of the examples for which solver i is the most effective (according to the statistic s_{ij}), $p_i(2)$ gives the fraction for which it is within a factor of 2 of the best, and $\lim_{\alpha \longrightarrow \infty} p_i(\alpha)$ gives the fraction for which the algorithm succeeded.

In this study, the statistics used are the CPU times required to perform the different phases of the solver, the number of nonzero entries in the matrix factor, and the total memory used by the solver (but in this paper, limitations on space allow us only to present CPU timings). Since some of the solvers we are examining are specifically designed for positive-definite problems (and may be unreliable, or even fail, on indefinite ones), we present our findings for the positive-definite and indefinite cases separately. In our tests, default values are used for all control parameters.

3 Preliminary Results

The numerical results were obtained on a Compaq DS20 Alpha server with a pair of EV6 CPUs; in our experiments only a single processor with 3.6 GBytes of RAM was used. The codes were compiled with full optimisation; the vendor-supplied BLAS were used. All CPU reported times are in seconds. A CPU limit of 2 hours was imposed for each code on each problem; any code that had not completed after this time was recorded as having failed. The scaled residual

$$\|b - Ax\|/(\|A\|\|x\| + \|b\|)$$

of each computed solution was checked before and after one step of iterative refinement; a residual after iterative refinement greater than 0.0001 causes an error to be flagged. In the reported tests, the input matrix A was not scaled.

3.1 Positive Definite Problems

The reliability of all the solvers in the positive-definite case was generally high. Only problem audikw was not solved by any code, this example being one of the two largest – it is of order roughly 950 thousand, and involves some 39 million nonzeros; the solvers with no out-of-core facilities were not able to allocate sufficient memory while the CPU time limit was exceeded for the remaining solvers. We present the performance profile for the CPU time for the complete solution (that is, the CPU time for analysing, factorising and solving for a single right-hand side) for the solvers in Figure 1, with the profiles for the separate analyse, factorise, and solve times in Figures 2 to 4. We see that, with the exception of `SPOOLES` and `UMFPACK` (which is primarily designed for unsymmetric problems), there is little to choose between the solvers when comparing the complete solution time. Many of the solvers use the nested dissection ordering from the METIS package [8] to obtain the pivot sequence and so they record similar analyse times. The multiple minimum degree algorithm used by `SPRSBLKLLT` is notably faster while `WSMP`

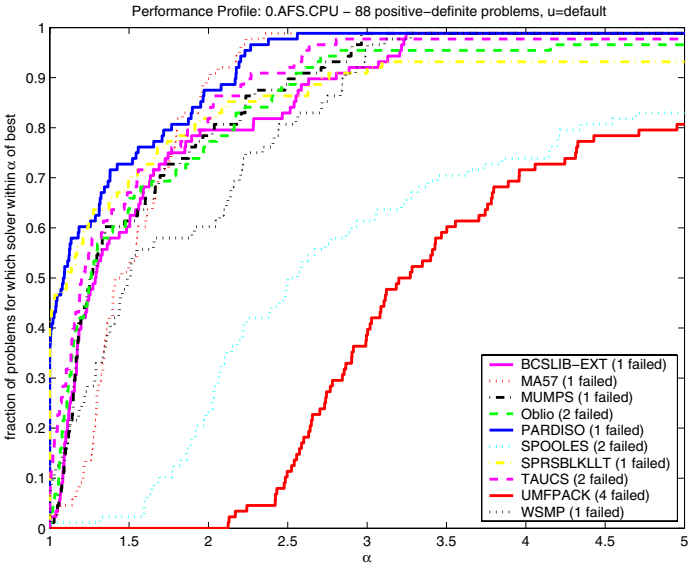

Fig. 1. Performance profile, $p(\alpha)$: CPU time (seconds) for the complete solution (positive-definite problems)

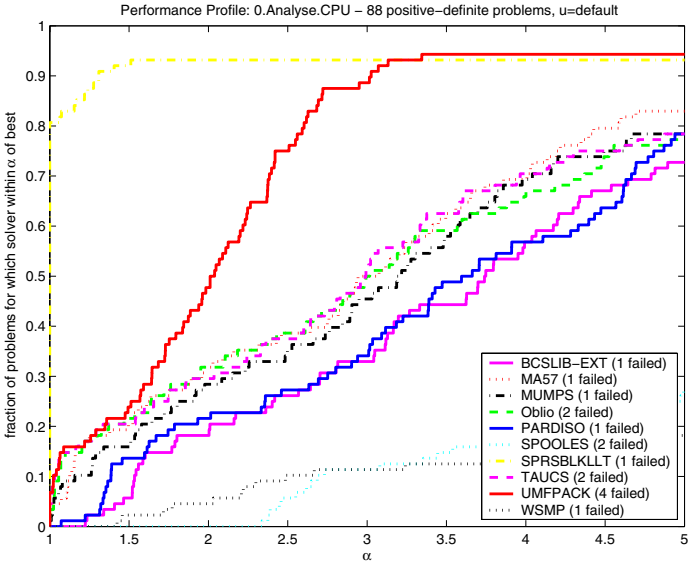

Fig. 2. Performance profile, $p(\alpha)$: CPU time (seconds) for analyse phase (positive-definite problems)

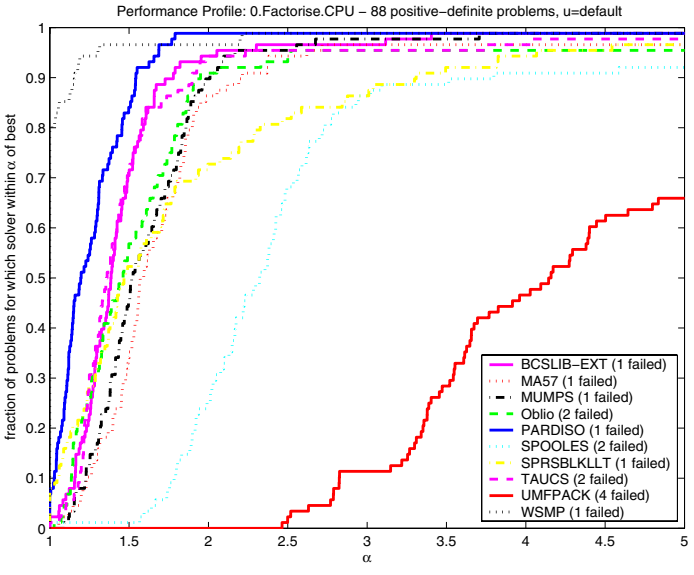

Fig. 3. Performance profile, $p(\alpha)$: CPU time (seconds) for the factorization (positive-definite problems)

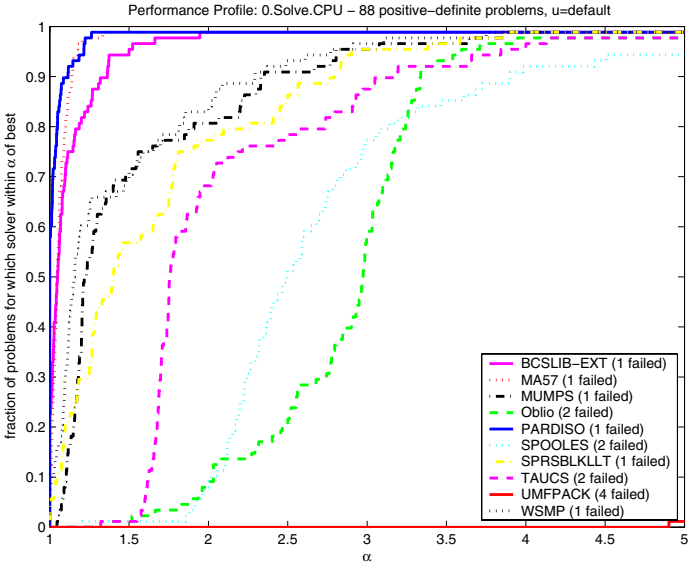

Fig. 4. Performance profile, $p(\alpha)$: CPU time (seconds) for the solve phase (positive-definite problems)

computes both a minimum local fill ordering and an ordering based on recursive bisection and selects the one that will result in the least fill-in. This extra investment pays dividends with WSMP having the fastest factorise times. In some applications, many solves may be required following the factorisation. The codes BCSLIB-EXT, MA57, and PARDISO have the fastest solve times.

3.2 Indefinite Problems

We now turn to the indefinite test suite. For these problems, pivoting is needed to maintain stability. The pivoting strategies offered by the codes are summarised in Table 3; more details are given in [5] and the references therein. Since SPRSBLKLLT and the tested version of TAUCS are designed only for solving definite problems, they are omitted. We have experienced problems when using SPOOLES for some indefinite systems, and thus results for SPOOLES are not currently included. MUMPS uses 1×1 pivots chosen from the diagonal and the factorization terminates if all the remaining (uneliminated) diagonal entries are zero. Because this may mean some of our test problems are not solved, at the authors' suggestion, we run both the symmetric and unsymmetric versions of MUMPS when testing indefinite examples.

Table 3. Default pivoting strategies

BCSLIB-EXT	Numerical pivoting with 1×1 and 2×2 pivots.
MA57	Numerical pivoting with 1×1 and 2×2 pivots.
MUMPS	Numerical pivoting with 1×1 pivots.
Oblio	Numerical pivoting with 1×1 and 2×2 pivots.
PARDISO	Supernode Bunch-Kaufmann within diagonal blocks.
SPOOLES	Fast Bunch-Parlett.
UMFPACK	Partial pivoting with preference for diagonal pivots.
WSMP	No pivoting.

The profiles for the indefinite results are given in Figures 5 to 8. The overall reliability of the solvers in the indefinite case was not as high as for the positive-definite one, with all the codes failing on some of the test problems. Because it does not include pivoting, WSMP had the highest number of failures. The majority of failures for the other codes were due to insufficient memory or the CPU time limit being exceeded or, in the case of symmetric MUMPS, no suitable diagonal pivots (singular matrices). The main exception was PARDISO, which had the fastest factorization times, but for two problems the computed solutions were found to be inaccurate.

We note that v1.0.0 of MA57 uses an approximate minimum degree ordering and computing this is significantly faster than computing the METIS ordering; this accounts for the fast MA57 analyse times. MA57 also has the fastest solve times; the solve times for PARDISO are slower even though it produces the sparsest factors since by default its solve phase includes one step of iterative refinement. Overall, PARDISO was the fastest code on our set of indefinite problems.

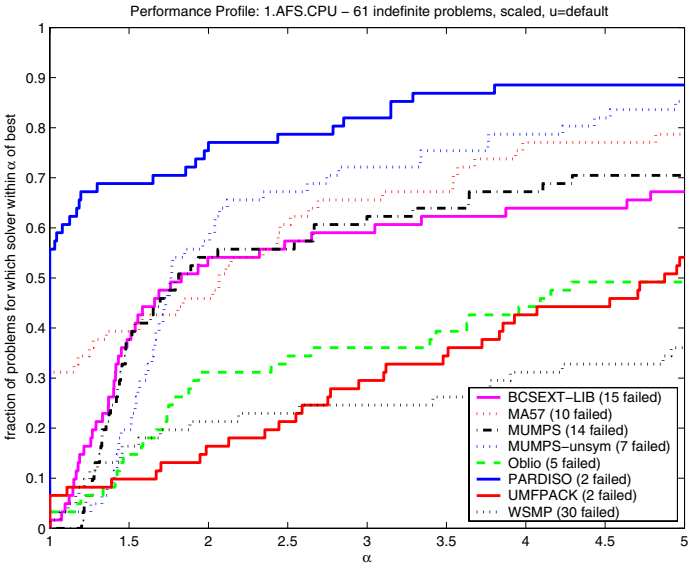

Fig. 5. Performance profile, $p(\alpha)$: CPU time (seconds) for the complete solution (indefinite problems)

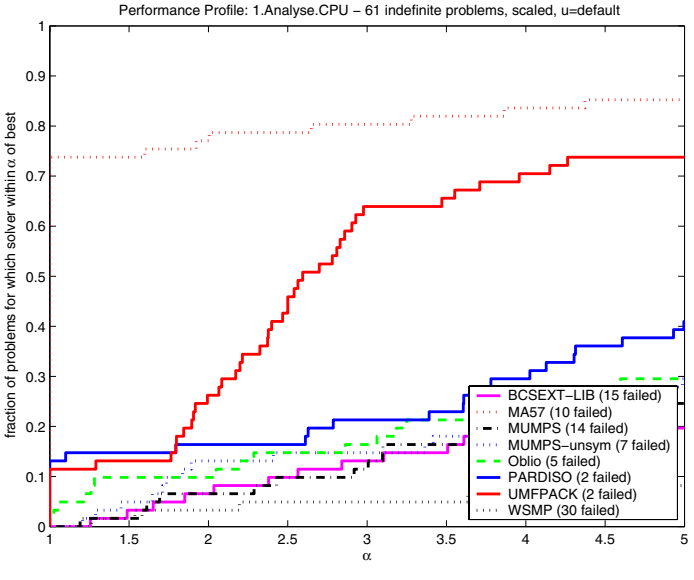

Fig. 6. Performance profile, $p(\alpha)$: CPU time (seconds) for analyse phase (indefinite problems)

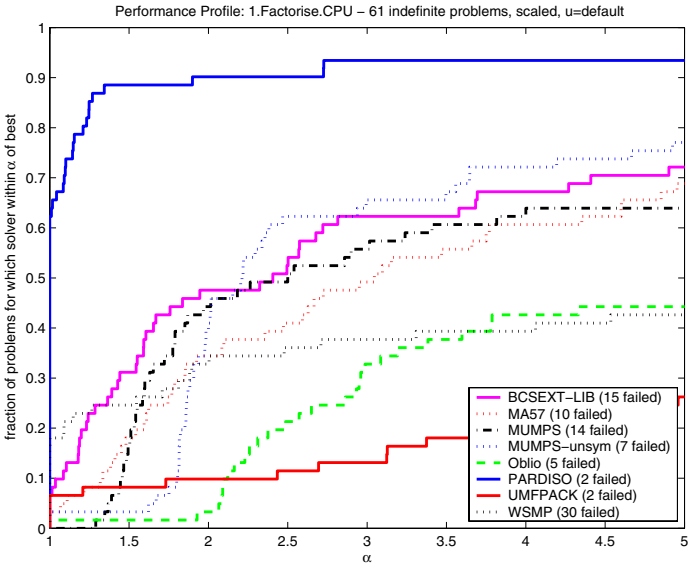

Fig. 7. Performance profile, $p(\alpha)$: CPU time (seconds) for the factorization (indefinite problems)

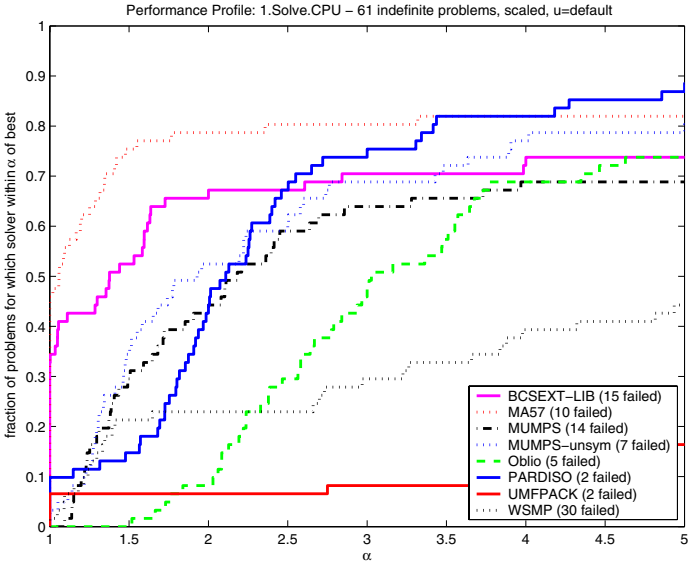

Fig. 8. Performance profile, $p(\alpha)$: CPU time (seconds) for the solve phase (indefinite problems)

4 General Remarks

In this paper, we have introduced our study of sparse direct solvers for symmetric linear systems and presented preliminary results. Full details of the study together with further

information on all the solvers used will be given in the forthcoming report [5]. This report will also contain more detailed numerical results and analysis of the results. Furthermore, since performance profiles can only provide a global view of the solvers, the full results for all the solvers on each of the test problems will be made available in a further report [6]. We note that our preliminary findings have already lead to modifications to a number of the solvers (notably `MA57`, `PARDISO`, and `Oblio`), and to further investigations and research into ordering and pivoting strategies.

Acknowledgements

We are grateful to all the authors of the solvers who supplied us with copies of their codes and documentation, helped us to use their software and answered our many queries. Our thanks also to all those who supplied test problems.

References

1. E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, **91**(2), 201–213, 2002.
2. I.S. Duff, R.G. Grimes, and J.G. Lewis. Sparse matrix test problems. *ACM Trans. Mathematical Software*, **15**, 1–14, 1989.
3. N.I.M. Gould and J.A. Scott. A numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations. *ACM Trans. Mathematical Software*, **30**, 300-325, 2004.
4. N.I.M. Gould and J.A. Scott. Complete results for a numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations. Numerical Analysis Internal Report 2003-2, Rutherford Appleton Laboratory, 2003. Available from www.numerical.rl.ac.uk/reports/reports.shtml.
5. N.I.M. Gould, Y. Hu and J.A. Scott. A numerical evaluation of sparse direct solvers for the solution of large, sparse, symmetric linear systems of equations. Technical Report, Rutherford Appleton Laboratory, 2005. To appear.
6. N.I.M. Gould, Y. Hu, and J.A. Scott. Complete results for a numerical evaluation of sparse direct solvers for the solution of large, sparse, symmetric linear systems of equations. Numerical Snalysis Internal Report, Rutherford Appleton Laboratory, 2005. To appear.
7. HSL. A collection of Fortran codes for large-scale scientific computation, 2002. See http://hsl.rl.ac.uk/.
8. G. Karypis and V. Kumar. METIS: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices - version 4.0, 1998. See http://www-users.cs.umn.edu/ karypis/metis/