Chapter 7

Iterative Solvers

This chapter describes the Krylov-based iterative solvers included in CMSSL. Section 7.2 provides references.

7.1 Krylov-Based Iterative Solvers

One important approach to solving large sparse linear systems is the use of iterative solvers based on Krylov subspace techniques. A well-known example of these techniques is the conjugate gradient (CG) method, which is used for symmetric positive definite systems. CG is a Lanczos-based method that reduces the problem matrix to a symmetric tridiagonal matrix. There are also Lanczos-based algorithms for non-symmetric systems; these entail the numerical problems associated with non-symmetric tridiagonal systems. Another class of non-symmetric algorithms is based on the Arnoldi procedure with its greater computational and storage requirements. Except for the Arnoldi-based restarted GMRES algorithm, all the algorithms currently included in CMSSL are Lanczos-based algorithms.

For pseudo-code and references for many of the non-symmetric Lanczos-based algorithms, see reference 1 listed in Section 7.2. References 2 and 3 also supply useful background.

7.1.1 CMSSL Iterative Solver Routines

Given a matrix A, a right-hand-side vector b, and a preconditioner $M = M_1{}^*M_2$ such that $A^{\sim} = M_1{}^{-1}AM_2{}^{-1}$, the **gen_iter_solve** routine (together with its associated setup and deallocation routines) solves the system $Ax =$

Copyright © 1994 Thinking Machines Corporation

b using Krylov space iterative methods. Any matrix operations (that is, matrix vector or vector matrix products) and preconditioning steps (for example, solve $My = z$) are provided by the user using a reverse communication interface. The nature of matrix and its internal representation are completely arbitrary, and depend on the user application. Similarly, the vectors can be represented by an array of any rank.

For details about the syntax, arguments, and usage of the iterative solvers, refer to the man page at the end of this section.

7.1.2 Algorithms

The iterative solvers offer the algorithms l
isted below. For detailed information about the algorithms, see the indicated references. (Full references are provided in Section 7.2.)

CMSSL_cg Conjugate gradient. A Lanczos-based algorithm for symmetric positive definite systems. (Note that this method will not work for non-symmetric systems.) See reference 4.

CMSSL_cgs Conjugate gradient squared of Sonneveld. See reference 5.

CMSSL_bcg Bi-conjugate gradient of Fletcher. See reference 6.

CMSSL_bicgstab Bi-conjugate gradient with stabilization of Van der Vorst. See reference 7.

CMSSL_bicgstab2 Bi-conjugate gradient with stabilization of Gutknecht. See reference 8.

CMSSL_gmres Restarted generalized minimal residual algorithm. See reference 9.

CMSSL_qmrcgs Transpose-free quasi-minimal residual (QMR) algorithm of Freund. See reference 10.

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

CMSSL_qmrlal	QMR with with a three-term look-ahead Lanczos algorithm of Freund, Gutknecht and Nachtigal. See references 11, 12, and 13.
CMSSL_qmr2	QMR based on two-term recursions for generating the Lanczos basis vectors without look-ahead. See reference 14.
CMSSL_qmrs	QMR squared of Freund and Szeto. See reference 15.
CMSSL_qmrbicgstab	QMR based on BICSTAB of Chan, Szeto and Tong. See reference 16.
CMSSL_qmrbicgstab2	A modified version of QMRBICGSTAB. See reference 16.
CMSSL_qcgs	Quasi-minimized CGS. See reference 1.

7.1.3 Acknowledgments

We wish to thank Roland Freund and Noel Nachtigal for providing us with the original Fortran 77 version of their code. We have converted their code to CM Fortran and included it in the algorithms avaliable in **gen_iter_solve**.

7.1.4 Example

The example below shows how to use the iterative solvers, and is based on the information in the man page. The application in this example provides three routines:

trid_matvec (*z, y, a, b, c, n*)	Multiplies Ay and places the results in z. A is represented by the diagonals ***a***, ***b***, and ***c***.
trid_vecmat (*z, y, a, b, c, n*)	Multiplies yA and places the results in z. A is represented by the diagonals ***a***, ***b***, and ***c***.
diag_solve (*z, y, M, n*)	Solves the system $Mz = y$.

Copyright © 1994 Thinking Machines Corporation

```
      call gen_iter_solve_setup(setup_iter,x,info,ier)

      ido = CMSSL_IDO_START
      do while ( (ido .ge. CMSSL_IDO_SOLVE_MIN) .and.
     &           (ido .le. CMSSL_IDO_SOLVE_MAX ) )

          call gen_iter_solve(ido,x,b,z,y,info,finfo,
     &            setup_iter,ier)

          reverse_comm: select case ( ido )

          case ( CMSSL_IDO_AY )
c
c         user supplied  z = A y
c
              call trid_matvec(z,y,a,b,c,n)

          case ( CMSSL_IDO_ATY )
c
c         user supplied  z = (A)T y
c
              call trid_vecmat(z,y,a,b,c,n)

          case ( CMSSL_IDO_SOLVE_M )
c
c         user supplied  z = (M)-1  y
c
              call diag_solve(z,y,M_1,n)

          case ( CMSSL_IDO_SOLVE_MT )
c
c         user supplied  z = ((M)T)-1  y
c
              call diag_solve(z,y,M_1,n)

          case ( CMSSL_IDO_SOLVE_M1 )
c
c         user supplied  z = (M1)-1  y
c
              call diag_solve(z,y,M1_1,n)

          case ( CMSSL_IDO_SOLVE_M2T )
c
c         user supplied  z = ((M2)T)-1  y
c
              call diag_solve(z,y,M2_1,n)

          case ( CMSSL_IDO_SOLVE_M2 )
c
```


Copyright © 1994 Thinking Machines Corporation

```
c          user supplied   z = (M2)-1   y
c
               call diag_solve(z,y,M2_1,n)

               end select reverse_comm

          enddo

          call deallocate_iter_solve(setup_iter)
```


Copyright © 1994 Thinking Machines Corporation

Iterative Solvers

Given a matrix A, a right-hand-side vector b, and a preconditioner $M = M_1{*}M_2$ such that $A^{\sim} = M_1{}^{-1}AM_2{}^{-1}$, the routines described below solve the system $Ax = b$ using Krylov space iterative methods. Any matrix operations (that is, matrix vector or vector matrix products) and preconditioning steps (for example, solve $My = z$) are provided by the user using a reverse communication interface. The nature of the matrix and its internal representation are completely arbitrary, and depend on the user application. Similarly, the vectors can be represented by an array of any rank.

SYNTAX

gen_iter_solve_setup (*setup, **x_template**, info, ier*)

gen_iter_solve (*ido, **x**, **b**, **z**, **y**, info, finfo, setup, ier*)

deallocate_iter_solve (*setup*)

ARGUMENTS

setup Scalar integer variable. Internal variable. The initial value you supply to **gen_iter_solve_setup** is ignored. When you call **gen_iter_solve** or **deallocate_iter_solve**, supply the value assigned to *setup* by the associated setup call. Do not change the value of *setup* after the setup routine returns.

x_template Real or complex (single- or double-precision) CM array with the same shape and layout as ***x***. Must be real if *info*(**CMSSL_iter_algorithm**) is **CMSSL_cg** or **CMSSL_bicgstab2**.

info Integer front-end array of rank 1 and length **CMSSL_iter_info_size**. When you call **gen_iter_solve_setup**, set *info* as indicated below. *Do not* change the values of *info* after the setup routine returns. Upon return from **gen_iter_solve**, *info*(**CMSSL_iter_liter**) contains the current (last) iteration step number; *info*(**CMSSL_iter_kspace_used**) contains the current size of the Lanczos subspace used in restarted GMRES or the current number of Lanczos vectors used by the look-ahead Lanczos algorithm (QMRLAL).

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

The values of the symbolic constants are defined in the CMSSL header file.

info(**CMSSL_iter_algorithm**)
Specifies the algorithm to be used. Supply one of the values listed below. For references, see Sections 7.1.2 and 7.2.

CMSSL_cg	Conjugate gradient. A Lanczos-based algorithm for symmetric positive definite systems. (Note that this method will not work for non-symmetric systems.)
CMSSL_cgs	Conjugate gradient squared of Sonneveld.
CMSSL_bcg	Bi-conjugate gradient of Fletcher.
CMSSL_bicgstab	Bi-conjugate gradient with stabilization of Van der Vorst.
CMSSL_bicgstab2	Bi-conjugate gradient with stabilization of Gutknecht.
CMSSL_gmres	Restarted generalized minimal residual algorithm.
CMSSL_qmrcgs	Transpose-free quasi-minimal residual (QMR) algorithm of Freund.
CMSSL_qmrlal	QMR with with a three-term look-ahead Lanczos algorithm of Freund, Gutknecht and Nachtigal.
CMSSL_qmr2	QMR based on two-term recursions for generating the Lanczos basis vectors without look-ahead.

Copyright © 1994 Thinking Machines Corporation

CMSSL_qmrs	QMR squared of Freund and Szeto.
CMSSL_qmrbicgstab	QMR based on BICSTAB of Chan, Szeto and Tong.
CMSSL_qmrbicgstab2	A modified version of QMRBICGSTAB.
CMSSL_qcgs	Quasi-minimized CGS.

info(**CMSSL_iter_init**)
Determines the contents of the initial guess, x_0. Set to 0 to specify $x_0 = 0$; set to 1 to use the initial input value of $\boldsymbol{x}$ for x_0.

info(**CMSSL_iter_random_start**)
Determines the initial residual value, r_0. Set to 0 to specify $r_0 = \boldsymbol{b} - Ax_0$; set to 1 to use a random value for r_0.

info(**CMSSL_iter_maxiter**)
Maximum number of iterations.

info(**CMSSL_iter_precond**)
Set to 1 for preconditioning, or 0 for no preconditioning. (See Description section for a discussion of preconditioning.)

info(**CMSSL_iter_kspace**)
If *info*(**CMSSL_iter_algorithm**) = **CMSSL_gmres**, this parameter specifies the maximum size of the Lanczos subspace used by GMRES (the maximum number of Lanczos vectors stored between restarts). If *info*(**CMSSL_iter_algorithm**) = **CMSSL_qmrlal**, this parameter specifies the maximum number of Lanczos vectors to look ahead on breakdown.

info(**CMSSL_iter_omega**)
Specifies the convergence criterion, ω. Set to 1 to use a convergence criterion of $\omega = ||\, r \,||_2 / ||\, \boldsymbol{b} \,||_2$, or set to 2 to use a convergence criterion of $\omega = ||\, r \,||_\infty / ||\, A \,||_\infty ||\, x \,||_1 + ||\, \boldsymbol{b} \,||_\infty$, where

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

$$r = \boldsymbol{b} - A\boldsymbol{x}$$
$$||\ r\ ||_n = (|r(1)|^n + |r(2)|^n + ...)^{1/n}$$
$$||\ r\ ||_\infty = \max\ |r(i)|$$

For **CMSSL_gmres**, the convergence criterion ω is set to the magnitude of the last Givens rotation used in reducing the upper Hessenberg matrix to upper triangular form.

info(**CMSSL_iter_residual**)
The value you supply is used only if you specified one of the following algorithms:

CMSSL_qmrcgs
CMSSL_qmr2
CMSSL_qmrs
CMSSL_qmrlal
CMSSL_qmrbicgstab
CMSSL_qmrbicgstab2
CMSSL_qcgs

If set to 0, this parameter causes **gen_iter_solve** to test an estimated residual for convergence against the convergence criterion, ω (see above) at the end of each iteration loop.

If you set this parameter to any value greater than 0, **gen_iter_solve** explicitly computes $r = \boldsymbol{b} - A\boldsymbol{x}$ every *info*(**CMSSL_iter_residual**) iterations. Thus, the extra matrix vector product needed to compute r can be amortized over several iterations. Convergence is only checked every *info*(**CMSSL_iter_residual**) iterations.

If *info*(**CMSSL_algorithm**) = **CMSSL_qmrlal**, $r = \boldsymbol{b} - A\boldsymbol{x}$ is explicitly computed every iteration when *info*(**CMSSL_residual**) is not equal to 0.

info(**CMSSL_iter_output_x**)
If you set this parameter to $m > 0$, $\boldsymbol{x}$ will contain intermediate values of the solution before convergence or breakdown every m iteration steps. If $m \geq$ *info*(**CMSSL_iter_maxiter**), $\boldsymbol{x}$ is guaranteed to contain the most recent value on return only when *ido* = **CMSSL_ido_end**, **CMSSL_ido_error**, or **CMSSL_ido_breakdown**.

Copyright © 1994 Thinking Machines Corporation

If you set *info*(**CMSSL_iter_output_x**) $\leq$ 0, it is reset to *info*(**CMSSL_iter_maxiter**). In this case, $\boldsymbol{x}$ is guaranteed to contain the most recent value on return only when *ido* = **CMSSL_ido_end**.

info(**CMSSL_iter_kspace_used**)
The input value is ignored. If *info*(**CMSSL_iter_algorithm**) = **CMSSL_gmres**, this parameter returns the size of the Lanczos subspace used by restarted GMRES during the current iteration loop.

If *info*(**CMSSL_iter_algorithm**) = **CMSSL_qmrlal**, this parameter returns the number of Lanczos vectors used during the look-ahead procedure to avoid breakdown during the current iteration loop.

info(**CMSSL_iter_liter**)
The input value is ignored. This parameter returns the current iteration loop count.

ido

Scalar integer variable used for reverse communication. The first time you call **gen_iter_solve** in a reverse communication loop, set *ido* = **CMSSL_ido_start**. On return, *ido* is set to one of the values listed below. All symbolic constants are defined in the CMSSL header file. M is the preconditioner, $M = M_1 M_2$, such that $A^{\sim} = {M_1}^{-1} A {M_2}^{-1}$.

CMSSL_ido_Ay
The user application must perform the matrix vector multiplication $A\boldsymbol{y}$ and place the results in $\boldsymbol{z}$.

CMSSL_ido_ATy
The user application must perform the matrix transpose vector multiplication, $A^{\mathrm{T}}\boldsymbol{y}$, or equivalently the vector matrix product, $\boldsymbol{y}^{\mathrm{T}}A$, and place the results in $\boldsymbol{z}$.

CMSSL_ido_solve_M
The user application must solve the system $M\boldsymbol{z} = \boldsymbol{y}$ (compute $M^{-1}\boldsymbol{y}$ and place the results in $\boldsymbol{z}$). If M is the identity I, replace $\boldsymbol{z}$ by the value in $\boldsymbol{y}$.

CMSSL_ido_solve_MT
The user application must solve the system $M^{\mathrm{T}}\boldsymbol{z} = \boldsymbol{y}$

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

(compute $M^{-T}y$ and place the results in z). If M is the identity I, replace z by the value in y.

CMSSL_ido_solve_M1

The user application must solve the system $M_1 z = y$ (compute ${M_1}^{-1}y$ and place the results in z). If M_1 is the identity I, replace z by the value in y.

CMSSL_ido_solve_M2T

The user application must solve the system ${M_2}^{T}z = y$ (compute ${M_2}^{-T}y$ and place the results in z). If M_2 is the identity I, replace z by the value in y.

CMSSL_ido_solve_M2

The user application must solve the system $M_2 z = y$ (compute ${M_2}^{-1}y$ and place the results in z). If M_2 is the identity I, replace z by the value in y.

CMSSL_ido_end

Convergence has occurred; that is, the convergence criterion ω is $\leq$ the initial input value of *finfo*(**CMSSL_iter_tol**).

CMSSL_ido_error

The maximum number of iterations specified in *info*(**CMSSL_iter_maxiter**) has been reached without convergence.

CMSSL_ido_breakdown

A breakdown in the algorithm (for example, division by 0) has occurred.

During the reverse communication loop, when **CMSSL_ido_solve_min** $\leq$ *ido* $\leq$ **CMSSL_ido_solve_max**, the algorithm is running normally and has not yet converged. If the value of *ido* falls outside this range, the algorithm has terminated without convergence (that is, either the maximum number of iterations has been reached or the algorithm has suffered a breakdown from which it cannot continue).

Copyright © 1994 Thinking Machines Corporation

x Real or complex (single- or double-precision) CM array of any rank and shape. Must be real if *info*(**CMSSL_iter_algorithm**) is **CMSSL_cg** or **CMSSL_bicgstab2**. On input, supply the initial guess to be used when *info*(**CMSSL_iter_init**) = 1. If you set *info*(**CMSSL_iter_output_x**) = $m > 0$, the intermediate value of the solution is returned in ***x*** every m iterations. When *ido* = **CMSSL_ido_end**, ***x*** contains the converged solution. When *ido* = **CMSSL_ido_error** or **CMSSL_ido_breakdown**, the value of ***x*** in the current iteration loop is returned.

b Real or complex (single- or double-precision) CM array with the same shape, layout, data type, and precision as ***x***. Must be real if *info*(**CMSSL_iter_algorithm**) is **CMSSL_cg** or **CMSSL_bicgstab2**. Contains the right-hand side of the system to be solved.

z Real or complex (single- or double-precision) CM array with the same shape, layout, data type, and precision as ***x***. Must be real if *info*(**CMSSL_iter_algorithm**) is **CMSSL_cg** or **CMSSL_bicgstab2**. Input argument; used only after the user application performs an operation that **gen_iter_solve** requested through reverse communication. Must contain the results of the user-supplied operation.

y Real or complex (single- or double-precision) CM array with the same shape, layout, data type, and precision as ***x***. Must be real if *info*(**CMSSL_iter_algorithm**) is **CMSSL_cg** or **CMSSL_bicgstab2**. Used only when **gen_iter_solve** returns with an *ido* value requesting the user application to operate on an array. Contains the array to be operated on by the user application.

finfo Real front-end array with rank 1, the same precision as ***x***, and length **CMSSL_iter_finfo_size** (a symbolic constant defined in the CMSSL header file). On input, set the values of *finfo* to supply the following information:

finfo(**CMSSL_iter_tol**)
On input, this parameter specifies the convergence tolerance, *tol*. If $\omega \leq tol$, **gen_iter_solve** returns with *ido* = **CMSSL_ido_end**, indicating that convergence has occurred.

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

On output, this parameter contains the value of ω computed for the current iteration loop.

finfo(**CMSSL_iter_anorm**)
Estimate of the infinity norm of A, $||A||_\infty$, to be used in computing the convergence criterion when *info*(**CMSSL_iter_omega**) = 2.

You can supply different values for these parameters each time you call **gen_iter_solve**.

ier Scalar integer variable. Error code. Upon return from **gen_iter_solve_setup**, contains one of the following values:

0 No error condition.

−**CMSSL_iter_algorithm**
info(**CMSSL_iter_algorithm**) is invalid.

−**CMSSL_iter_kspace**
info(**CMSSL_iter_kspace**) is invalid for *info*(**CMSSL_iter_algorithm**) = **CMSSL_gmres** or **CMSSL_qmrlal**.

−**CMSSL_iter_maxiter**
info(**CMSSL_iter_maxiter**) is < 1.

The *ier* argument to the **gen_iter_solve** routine is reserved for future use.

DESCRIPTION

Setup and Deallocation. To use the iterative solvers, follow these steps:

1. Call **gen_iter_solve_setup**.

   This routine generates a setup ID and returns it in the *setup* argument. You must supply this *setup* value in all subsequent **gen_iter_solve** and **deallocate_iter_solve** calls associated with this setup call.

2. Call **gen_iter_solve**. Supply the *setup* value assigned by the setup routine, and the same *info* values you supplied to **gen_iter_solve_setup**. In particular, different algorithms require different amounts of internal storage; so if you change the algorithm, call **deallocate_iter_solve** and then call **gen_iter_solve_setup** again. (This involves very little overhead compared to the rest of the algorithm).

Copyright © 1994 Thinking Machines Corporation

Each call to **gen_iter_solve** runs the specified algorithm until intermediate operations must be supplied by the user application; **gen_iter_solve** then requests these operations through the reverse communication interface.

3. When **gen_iter_solve** returns with an *ido* value requesting action by the user application, you must supply the requested operation, place the results in the array *z*, and call **gen_iter_solve** again. Continue until the returned *ido* value indicates convergence, maximum number of iterations exceeded, or breakdown.

4. You can solve other systems by repeating Steps 2 and 3, as long as the rank and shape of $\boldsymbol{x}$ remain the same and the input values you supplied in the setup routine's *info* argument still apply. If the rank and shape of $\boldsymbol{x}$ change or you need to change any of the *info* values, start with Step 1 again.

5. After all **gen_iter_solve** calls associated with the same call to **gen_iter_solve_setup**, call **deallocate_iter_solve** to deallocate the memory required by the setup routine.

More than one setup may be active at a time, as long as they use different setup parameters. That is, you may call the setup routine more than once without calling the deallocation routine.

Iteration. The **gen_iter_solve** routine attempts to solve $A\boldsymbol{x} = \boldsymbol{b}$ using one of several Krylov space iterative solution algorithms. Depending on the algorithm and the value of the parameter *info*(**CMSSL_iter_residual**), either the actual residual $r = \boldsymbol{b} - A\boldsymbol{x}$ or an estimate of the residual is used to compute a convergence parameter ω which is returned in *finfo*(**CMSSL_iter_tol**) at each step of the iteration. The iterations continue until $\omega \leq$ the initial input value of *finfo*(**CMSSL_iter_tol**), an algorithmic breakdown has occurred, or the number of iterations has exceeded the value supplied in *info*(**CMSSL_iter_maxiter**).

Reverse Communication Interface. The iterative solvers require the user application to provide

- routines that multiply a given vector $\boldsymbol{y}$ by A or A^{T} (alternatively, a vector matrix multiplication, $A^{\mathrm{T}}\boldsymbol{y} = (\boldsymbol{y}^{\mathrm{T}}A)^{\mathrm{T}}$)

- routines that solve the systems $M\boldsymbol{z} = \boldsymbol{y}$, $M^{\mathrm{T}}\boldsymbol{z} = \boldsymbol{y}$, $M_1\boldsymbol{z} = \boldsymbol{y}$, $M_2{}^{\mathrm{T}}\boldsymbol{z} = \boldsymbol{y}$, and $M_2\boldsymbol{z} = \boldsymbol{y}$, where $M = M_1M_2$ and the preconditioned matrix $\tilde{A} = M_1{}^{-1}AM_2{}^{-1}$

The vectors can have any shape. When **gen_iter_solve** requires one of these user-supplied operations, it returns, sets *ido* to indicate which operation is required, and

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

provides the vector *y* upon which the user application is to operate. The user-supplied routine must place the results of the requested operation in *z* and call **gen_iter_solve** again.

Preconditioning. If you set *info*(**CMSSL_iter_precond**) = 1, **gen_iter_solve** asks you to solve systems involving the preconditioner, $M = M_1 M_2$, where the preconditioned system is given by $\tilde{A} = {M_1}^{-1} A \, {M_2}^{-1}$, $\tilde{A} M_2 \boldsymbol{x} = {M_1}^{-1} \boldsymbol{b}$.

NOTES

Include the CMSSL Header File. The iterative solvers use symbolic constants defined in the CMSSL header file. Therefore, you must include the line

```
INCLUDE '/usr/include/cm/cmssl-cmf.h'
```

at the top of any program module that calls these routines.

Acknowledgments. We wish to thank Roland Freund and Noel Nachtigal for providing us with the original Fortran 77 version of their code. We have converted their code to CM Fortran and included it in the algorithms available in **gen_iter_solve**.

EXAMPLES

Sample CM Fortran code that uses the iterative solvers can be found on-line in the subdirectory

iter-solvers/cmf

of a CMSSL examples directory whose location is site-specific.

Copyright © 1994 Thinking Machines Corporation

7.2 References

For more information about iterative solvers, see the following references:

1. Tong, C. H. *A Comparative Study of Preconditioned Lanczos Methods for Non-symmetric Linear Sytems*. Sandia Technical Report SAND91-8240, 1992.

2. Nachtigal, N.M., S.C.Reddy, and L.N. Trefethen. *How Fast Are Non-Symmetric Matrix Iterations?* Dept. of Mathematics, MIT, Numerical Analysis Report 90-2, 1990.

3. Freund, R.W., G. H. Golub, and N. M. Nachtigal. *Iterative Solution of Linear Systems*. Preprint.

4. Golub, G.H. and C.F. Van Loan. *Matrix Computations*. 2d. ed. Baltimore: Johns Hopkins University Press, 1989.

5. Sonneveld, P. CGS, a Fast Lanczos-Type Solver for Non-Symmetric Linear Systems. *SIAM J. Sci. Stat. Comput.* **10** (1989): 36−52.

6. Fletcher, R. Conjugate Gradient Methods for Indefinite Systems. In *Proc.*
*Dundee Confere*nce on Numerical Analysis*, 1975, Lecture Notes in Mathematics 506*, ed. G.A. Watson. Berlin: Springer-Verlag, 1976. Pp. 73−89.

7. Van der Vorst, H.A. BI-CGSTAB: A Fast and Smoothly Converging Variant of BI−CG for the Solution of Non-Symmetric Linear Systems. *SIAM J. Sci. Stat. Comput.* **13** (1992): 631−644.

8. Gutknecht, M.H. *Variants of BICGSTAB for Matrices with Complex Spectrum*. IPS Research Report No. 91-14, Interdisciplinary Project Center for
Supercomputing. ETH-Zetrum, CH-8092, Zurich, 1991.

9. Saad, Y. and M.H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Non-Symmetric Linear Systems. *SIAM J. Sci. Stat. Comput.* **7** (1986): 856−869.

10. Freund, R.W. *A Transpose-Free Quasi-Minimal Residual Algorithm for Non-Hermitian Linear Systems*. RIACS Technical Report 91.18, NASA Ames Research Center, Moffett Field, California.

Copyright © 1994 Thinking Machines Corporation

11. Freund, R. W., M.H. Gutknecht, and N.M. Nachtigal. An Implementation of the Look-Ahead Lanczos Algorithm for Non-Hermitian Matrices. To appear in *SIAM J. Sci. Stat. Comput.* **14** (1993).

12. Freund, R.W. and N.M. Nachtigal. *An Implementation of the Look-Ahead Lanczos Algorithm for Non-Hermitian Matrices, Part II*. RIACS Technical Report 90.46, NASA Ames Research Center, Moffett Field, CA, 1990.

13. Freund, R.W. and N.M. Nachtigal. *QMR:* A Quasi-Minimal Residual Method for Non-Hermitian Linear Systems. *Numer. Math.* **60** (1991): 315−39.

14. Freund, R. W. and N.M. Nachtigal. *An Implementation of the QMR Method Based on Coupled Two-Term Recurrences*. RIACS Technical Report 92.15, NASA Ames Research Center, Moffett Field, CA, 1992.

15. Freund, R.W. and T. Szeto. A Quasi-Minimal Residual Squared Algorithm for Non-Hermitian Linear Systems. In *Proceedings of the 1992 Copper Mountain Conference on Iterative Methods*, 1992.

16. Chan, T. F., E. Gallopoulos, V .Simoncini, T. Szeto, and C. H. Tong. *QMRCGSTAB: A Quasi-Minimal Residual Variant of the BI-CGSTAB Algorithm for Non-Symmetric Systems.* CSRD Technical Report 1231, University of Illinois at Urbana-Champaign, Illinois, 1992.

Copyright © 1994 Thinking Machines Corporation