CMSSL 3.2 Release Notes for CM Fortran

1 Introduction

These release notes summarize the changes and new features in the CM Fortran interface to Version 3.2 of the CM Scientific Software Library (CMSSL). One section is devoted to each of the following topics:

- Hardware and software requirements
- New features
- Changes from previous versions
- Limitations and restrictions
- Documentation for this release
- Acknowledgments

CMSSL Version 3.2 includes all the functionality of Version 3.1, as well as new functionality described in Section 3.

2 Hardware and Software Requirements

2.1 Hardware Required

CMSSL Version 3.2 supports CM-5 systems with or without vector units, and supports the nodal CMSSL library. You can also call this version of CMSSL from global/local CM Fortran programs. The nodal and global/local execution models require a CM-5 with vector units.

Copyright © 1994 Thinking Machines Corporation

2.2 Software Required

CMSSL Version 3.2 for CM Fortran requires prior installation of the following software:

- CMOST Version 7.2 (or higher)
- CM Fortran Version 2.1 (or higher)

In addition, for CM Fortran programs that run in the single-node execution model (**-node**), you must install CMMD Version 3.0 (or higher). If the default CMMD is not an appropriate version, then you may need to use

```
-cmmd_root /usr/cmmd/3.0
```

in your link line.

2.3 Linking with CMSSL Version 3.2

After writing a CM Fortran program that calls CMSSL routines, compile it and link it with the library. Compiling a CM Fortran CMSSL program is the same as compiling other CM Fortran programs: use the **cmf** command. To compile the program *program* on a CM-5 and link it with CMSSL, you can use either of the alternatives indicated below.

Linking with CMSSL Using −lcmssl

If the CM Fortran program *program* calls CMSSL routines, you can compile it and link it with CMSSL using the **-lcmssl** switch. This switch links with the appropriate CMSSL library based on the execution modal you specify in the **cmf** command line. To use this alternative, issue one of the following command lines at the UNIX prompt:

- In the CM Fortran (SPARC) nodes model:

  cmf -cm5 -sparc -o *program program*.**fcm -lcmssl**

- In the CM Fortran vector-units model:

  cmf -cm5 -vu -o *program program*.**fcm -lcmssl**

- In the CM Fortran single-node model:

  cmf -cm5 -vu -node -o *program program*.**fcm -lcmssl**

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

In the single-node model, add **-cmmd_root /usr/cmmd/3.0** before **-lcmsslcm5vu-node** if CMMD Version 3.0 (or higher) is not the default on your system.

- To compile and link a CM Fortran global/local program that makes calls to CMSSL routines in both the global and the local program units:

  cmf -cm5 -vu -o *program program*.**fcm -lcmssl -local** *local*.**fcm** *program*.**proto -local -lcmssl**

Linking with a Selected CMSSL Library Explicitly

Alternatively, if you want to link with the appropriate CMSSL library explicitly, issue one of the following command lines at the UNIX prompt:

- In the CM Fortran (SPARC) nodes model:

  cmf -cm5 -sparc -o *program program*.**fcm -lcmsslcm5**

- In the CM Fortran vector-units model:

  cmf -cm5 -vu -o *program program*.**fcm -lcmsslcm5vu**

- In the CM Fortran single-node model:

  cmf -cm5 -vu -node -o *program program*.**fcm -lcmsslcm5vu-node**

  In the CM Fortran single-node model, add **-cmmd_root /usr/cmmd/3.0** before **-lcmsslcm5vu-node** if CMMD Version 3.0 (or higher) is not the default on your system.

- To compile and link a CM Fortran global/local program that makes calls to CMSSL routines in both the global and the local program units:

  cmf -cm5 -vu -o *program program*.**fcm -lcmsslcm5vu -local** *local*.**fcm** *example*.**proto -local -lcmsslcm5vu-node**

Copyright © 1994 Thinking Machines Corporation

3 New Features

The CM Fortran operations listed below are new since Version 3.1. All chapter numbers refer to *CMSSL for CM Fortran*, Version 3.2.

- *Gaussian elimination with external storage*

  The external *LU* factorization and solver routines have been completely rewritten and optimized in Version 3.2; they have a new interface that involves reverse communication. (Chapter 5)

- *Fixed-machine-size save and restore routines for LU and QR solvers*

  The *LU* and *QR* routines are joined by new save and restore routines: **save_gen_lu_fms**, **restore_gen_lu_fms**, **save_gen_qr_fms**, and **restore_gen_qr_fms**. These new routines have the same calling sequences and perform the same functions as the corresponding older routines (**save_gen_lu**, **restore_gen_lu**, **save_gen_qr**, and **restore_gen_qr**, respectively), except that the new routines use fixed-machine-size I/O while the older ones use serial-order I/O. (Chapter 5)

- *Other new Gaussian elimination routines*

  The Gaussian elimination routines described in Chapter 5 of *CMSSL for CM Fortran* are joined by three new routines in Version 3.2: **gen_lu_apply_l**, **gen_lu_apply_l_tra**, and **gen_lu_zero_rows**. (Chapter 5)

- *Bidiagonalization*

  Version 3.2 introduces the **gen_bidiag** routine, which transforms dense real matrices to bidiagonal form. Auxiliary routines transform arbitrary vectors between the basis of the original dense matrix and the basis of the bidiagonal matrix. (Chapter 9)

- *Singular values of a bidiagonal matrix*

  The **bidiag_svd_singular_values** routine computes all the singular values of one or more real bidiagonal matrices. (Chapter 9)

- *Singular vectors of a bidiagonal matrix*

  The **bidiag_svd_singular_vectors** routine computes the singular vectors corresponding to a set of singular values of one or more real bidiagonal matrices of the same order. (Chapter 9)

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

- *Singular value decomposition of dense real matrices*

  The **gen_bidiag_singular_system** routine performs a singular value decomposition of one or more dense real matrices. (Chapter 9)

- *Range histogram with CM array of bins*

  Version 3.2 introduces a new histogram routine, **histogram_range_cm**, that performs the same function as the **histogram_range** routine, but stores the bins in a CM array rather than a front-end array. (Chapter 14)

- *Combination of permutations*

  The new **combine_fe_perms** routine returns the combination of two permutations supplied to it in front-end arrays. (Chapter 15)

- *Zeroing routine*

  This release introduces a new routine, **zero_elements**, that zeroes a CM array. This routine provides faster performance than the equivalent CM Fortran code, especially for single-precision real or complex data. (Chapter 15)

4 Changes from Previous Versions

The following CM Fortran routines have changed since Version 3.1:

- *Arbitrary elementwise sparse matrix operations*

  The *irandom*, *itrace*, and *trace* arguments to the arbitrary elementwise sparse matrix operations have been renamed to *mapping*, *motion*, and *setup*, respectively. The *mapping* and *motion* arguments provide new options for permutation method and communication algorithm, respectively. The *mapping* argument now makes available source and destination array permutations generated internally using the **partition_mesh** routine, for problems with symmetric sparsity. The *motion* argument provides an option for communication that uses the **part_gather** and **part_scatter** routines, as an alternative to the **sparse_util_gather** and **sparse_util_scatter** routines. The values accepted by these arguments are as follows:

Copyright © 1994 Thinking Machines Corporation

Value of *mapping*	Permutation returned in ***where_is_x*** and ***where_is_y***
0	identity (old *irandom* = 0)
1	random (old *irandom* = 1)
2	based on partitioning

Value of *motion*	Operations Used by Communication algorithm
0	get, send, scan-add (old *itrace* = 0)
1	**part_gather**, **part_scatter**
2	**sparse_util_gather**, **sparse_util_scatter** (old *itrace* = 1)

Additionally, in the Version 3.1 manual, the man page for these routines had placed the ***where_is_x*** and ***where_is_y*** arguments in the wrong order; they have been switched in the Version 3.2 man page. (Chapter 4)

- *Arbitrary elementwise sparse matrix operations*

  The *irandom, itrace, trace*, *trace_mask*, and *setup* arguments of the arbitrary block sparse matrix routines have been changed to *mapping, motion, setup1*, *setup2,* and *setup3*, respectively. Permutations based on partitioning are not yet available with these routines, but the *motion* values are the same as above (except that *motion* = 0 uses gets and send-adds). (Chapter 4)

- *QR and LU Routines*

  The *QR* routines now allow *nblock* values greater than 1 with pivoting. The *LU* routines now allow you to specify $m > n$ in the no-pivoting case. (However, you may not use the **gen_lu_get_l** routine when $m > n$.) The *LU* and *QR* factors are now clearly defined in the manual for the non-square case. (Chapter 5)

- *Range histogram*

  Beginning with Version 3.2, the **histogram_range** and **histogram_range_cm** routines require the *min* and *range* arguments to have the same data type (integer or real) as the destination array, ***A***. (Chapter 14)

- *Banded System Solvers*

  The *nblock* argument of the **gen_banded_factor**, **block_tridiag_factor**, **block_tridiag_solve**, **block_pentadiag_factor**, and **block_pentadiag_solve**

Copyright © 1994 Thinking Machines Corporation

routines has a new name and meaning. The argument, now called *group_size*, allows you to specify (or ask the routine to select) the number of problem instances per processing element that are treated together in one step of Gaussian elimination. This feature can significantly improve performance in comparison to CMSSL Version 3.1. In addition, the *work* argument is now a scalar integer rather than a front-end array. (Chapter 6)

- *Iterative Solvers*

  The iterative solvers now support complex data for all algorithms except **CMSSL_cg** (which is used for symmetric positive definite systems) and **CMSSL_bicgstab2**. Previous to Version 3.2, all algorithms required real data. (Chapater 7)

- *Generalized Eigensystem Analysis*

  The **sym_tred_gen_eigensystem** routine now operates on complex Hermitian matrices; previously it operated only on real symmetric matrices. (Chapter 8)

- *Simplex routine*

  The **gen_simplex** routine now checks the value of *ier* on input. If *ier* is set to 2 (reinvert) or 16 (degenerate), **gen_simplex** assumes you are reinverting; otherwise, it assumes you are passing it a new problem.

  Two new performance guidelines take effect with Version 3.2. These new guidelines, together with the two conditions already listed in the **gen_simplex** man page, ensure that **gen_simplex** does not copy the CM array ***A***:

  - Lay out ***A*** so that the subgrid size of the first axis is not 1.
  - Compile the main program with **-axisreorder**.

  (Chapter 12)

- *Matrix transpose performance enhancements*

  The axis-length restriction that was imposed by the **gen_matrix_transpose** routine in order to achieve enhanced performance has been lifted in Version 3.2. In addition, Version 3.2 adds another enhancement for three-dimensional and higher-rank arrays. The **gen_matrix_transpose** routine yields superior performance when a local axis is exchanged with a non-local axis that is distributed across the vector units on the same processing node. By far the best performance

Copyright © 1994 Thinking Machines Corporation

is obtained when the non-local axis spans only two vector units (that is, only the lowest off-chip bit is set). The next best performance results when the non-local axis spans four vector units on the same processing node (the two lowest off-chip bits are set). For the general case of an axis that spans multiple processing nodes, the transpose performance does not depend significantly of the number of contiguous off-chip bits (except if the axis spans only two nodes, in which case the performance is slightly better). You can use this fact to improve the performance of transposes involving arrays of rank greater than or equal to three. With nodal CMSSL, you can also exploit this fact with two-dimensional arrays. (Chapter 15)

- *Fast Fourier Transform*

  The **fft_setup** and **fft** routines now perform error checking; see Chapter 10 of *CMSSL for CM Fortran* for details. In addition, the enhancements to **gen_matrix_transpose** mentioned above may help users who perform transposes explicitly when performing FFTs on multidimensional arrays. (Chapter 10)

- *Sparse gather, scatter, vector gather, and vector scatter utilities*

  In the sparse vector gather and scatter utilities, the vectors are no longer required to lie along the left-most axis; you choose the vector axes in the source and destination arrays using two new arguments, *x_vector_axis* and *y_vector_axis*.

  In addition, the calling sequences of these routines now more closely resemble those of the partitioned gather and scatter utilities. The following changes have been made in Version 3.2:

  - The *trace* and *trace_mask* arguments of **sparse_util_gather** and its associated routines were combined into a single *setup* argument.
  - The ***y_template*** and ***x_mask*** arguments exchanged places in **sparse_util_scatter_setup** and **sparse_util_scatter_setup**.
  - The error code argument *ier* was added to **sparse_util_scatter_setup** and **sparse_util_vec_scatter_setup**.
  - The pointers argument ***p*** was removed from **sparse_util_scatter**.

  (Chapater 15)

Copyright © 1994 Thinking Machines Corporation

- *Partitioning*

  The following new features were introduced into the **partition_mesh** routine in Version 3.2:

  - You can now divide a mesh into partitions that contain multiple processing elements. One use for this feature would be to divide a very large mesh (which cannot be handled as one piece by your application) into several smaller pieces that can be processed sequentially. The *numproc* argument is now an input as well as output argument; it helps determine the number of processing elements in each partition.

  - A new argument, *storage_option*, provides a low-storage option that is slower than the default operation, but uses less storage for working arrays.

  - A new argument, *verbose*, prints statistics to standard output.

  In addition, in Version 3.2, you can choose the axis along which to reorder pointers with the **reorder_pointers** routine. Previously, this routine always reordered a pointers array along its last axis. You can also choose which axis counts the mesh elements in the ***ien*** and ***idual*** arrays. Moreover, a new element type (segment) has been added. (Chapter 15)

- *Partitioned gather and scatter utilities*

  The **part_gather**, **part_scatter**, **part_vector_gather**, **part_vector_scatter**, and associated routines have changed in the following ways in Version 3.2:

  - The *trace* argument was renamed *setup*.

  - The routines now take advantage of data locality along any axis; the reordered axis need not be the last axis. For best performance, the reordered axis should be non-local and all other axes should be local. (A *local axis* is either **:serial** or laid out with **:procs** = 1.)

  (Chapter 15)

- *Optimization Hints for All-to-All Broadcast and Reduction*

  The performance guidelines for the all-to-all broadcast and reduction routines have been updated to take into account the CM Fortran **-noaxisreorder** switch. (Chapter 15)

Copyright © 1994 Thinking Machines Corporation

- *Vector move (extract and deposit)*

  The vector move routine has a new error code. Additionally, the description of the vector move operation in *CMSSL for CM Fortran, CM-5 Edition*, Version 3.1, was incorrect. The vector move routine moves one vector from each subgrid of the source CM array into a subgrid of the destination CM array. (Chapter 15)

5 Limitations and Restrictions

The routines listed below cannot be used with the nodal CMSSL library (the library used when you compile your program in the CM Fortran single-node execution model). Also, if you are using the CM Fortran global/local execution model, do not call these routines from the local program unit.

save_gen_lu	**gen_lu_solve_ext**	**gen_qr_solve_ext**
restore_gen_lu	**save_gen_qr**	**gen_matrix_mult_ext**
save_gen_lu_fms	**restore_gen_qr**	**save_fast_rng_temps**
restore_gen_lu_fms	**save_gen_qr_fms**	**restore_fast_rng_temps**
gen_lu_setup_ext	**restore_gen_qr_fms**	**save_vp_rng_temps**
gen_lu_factor_ext	**gen_qr_factor_ext**	**restore_vp_rng_temps**

6 Documentation for This Release

The CM Fortran interface to CMSSL Version 3.2 is documented in *CMSSL for Fortran*, Version 3.2. The information that is summarized in these release notes is presented in more detail in the manual.

Your software tape includes ASCII and PostScript versions of these release notes and of *CMSSL for CM Fortran*. The default location for the release notes is

```
/usr/doc/cmssl-3.2-releasenotes
```

The default location for *CMSSL for CM Fortran,* Version 3.2 is

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

/usr/doc/cmssl/cmssl-for-cmf-v1-3.2 (Volume I)
/usr/doc/cmssl/cmssl-for-cmf-v2-3.2 (Volume II)

Within each volume directory you will find a PostScript file for each chapter. The file called **README** contains more information.

If you do not find the documents in the default locations, check with your system administrator or the person who installs the CMSSL at your site.

6.1 On-Line Sample Code and Man Pages

Included with CMSSL are sample on-line programs that demonstrate how to call each CMSSL routine. You are encouraged to experiment with these sample programs. Also included with CMSSL are on-line man pages for all routines.

The on-line sample programs are located in subdirectories of the CMSSL examples directory. The default location for the examples directory is

/usr/examples/cmssl.

Examples for the operation *operation* are included in the subdirectory

*operation***/cmf** (or *operation*/*sub-operation***/cmf**)

or

*operation***/cstar** (or *operation*/*sub-operation***/cstar**)

of the examples directory. For example, CM Fortran sample code for the routine that performs eigenvector analysis using the Jacobi method is located in the subdirectory

eigen/jacobi/cmf

of the examples directory. If you do not find the on-line examples in **/usr/examples/cmssl**, check with your system administrator (or the person who installs the CMSSL at your site) to find out where they were installed.

To read the on-line man page for a routine, enter the command

Copyright © 1994 Thinking Machines Corporation

man *routine_name*

at the UNIX prompt.

7 Acknowledgments

The bidiagonalization and singular value decomposition routines introduced in this release are the result of collaborative development between Thinking Machines Corporation and the Danish Computing Center for Research and Education (UNI−C).

Copyright © 1994 Thinking Machines Corporation