Chapter 12

CM Scientific Software Library

The Connection Machine Scientific Software Library (CMSSL) is a rapidly growing set of numerical routines that support computational applications while exploiting the massive parallelism of the Connection Machine system. The CMSSL provides data parallel implementations of familiar numerical routines, offering new solutions for performance optimization, algorithm choice, and application design. CMSSL routines have been designed for users of languages with array syntax (for example, CM Fortran, High Performance Fortran, and C*).

The CMSSL includes routines for solving linear algebraic equations, solving ordinary and partial differential equations, signal processing, statistical analysis, and optimization. The library also provides a set of communication functions that offer a strong base for the development of computational tools. These functions support computations on problems represented by both structured and unstructured grids. For computations on unstructured grids, the CMSSL offers routines for efficient load balancing of both arithmetic and communication.

12.1 Overview

The current version of the CMSSL concentrates on six critical areas of scientific programming:

- numerical linear algebra
  - matrix operations on dense, grid sparse, and arbitrary sparse matrices
  - linear equation solvers for dense, banded, and sparse systems of equations

Copyright © 1993 Thinking Machines Corporation

    - eigensystem analysis of dense symmetric, tridiagonal, and sparse systems
- Fourier Transforms (complex-to-complex, real-to-complex, and complex-to-real)
- ordinary differential equations
- optimization
- random number generation
- statistical analysis

The library also includes optimized communication functions important to structured and unstructured grid computations for the solution of partial differential equations and optimization problems:

- polyshift
- all-to-all broadcast and reduction
- matrix transpose
- gather and scatter
- partitioning
- communication compiler

12.2 Multiple Instances

Most CMSSL linear algebra routines are designed to support multiple instances. They allow multiple independent matrices to be solved, transformed, or multiplied concurrently. In addition, they allow multiple vectors or multiple right-hand sides, where relevant, to be associated with each matrix to be multiplied or solved. The difference between invoking computation on a single instance and on multiple instances lies only in the dimensionality and layout of the data structures used as parameters to the particular CMSSL routine.

As an example, consider the linear equation solvers for banded systems. For the tridiagonal case, the parameters to these routines include three vectors that contain the upper, main, and lower diagonals of a tridiagonal system, and a fourth vector that contains the right-hand-side values for the system. Upon completion

November 1993
Copyright © 1993 Thinking Machines Corporation

the solution overwrites the right-hand side. One routine interface supports four different degrees of computational concurrency:

- A single system may be solved.
- A single system may be solved for multiple right-hand sides.
- Multiple systems may be solved for a single right-hand side each.
- Multiple systems may be solved, each for multiple right-hand sides.

To solve a single system, one specifies the upper, main, and lower diagonal arguments as one-dimensional (see Figure 19).

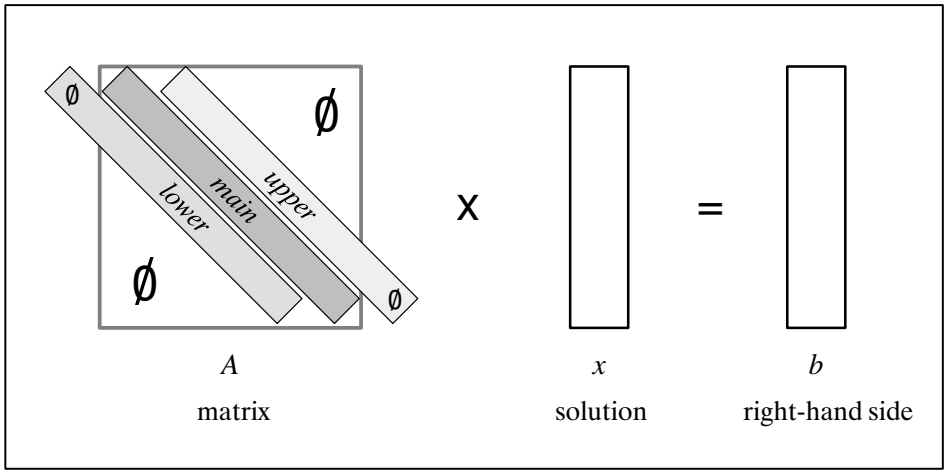

Figure 19. A single tridiagonal system with a single right-hand side.

To solve for multiple right-hand sides, one gives the right-hand-side argument (which will be replaced by the solutions) an in-processor (serial) dimension equal to the number of right-hand sides (*nrhs*) (see Figure 20).

Copyright © 1993 Thinking Machines Corporation

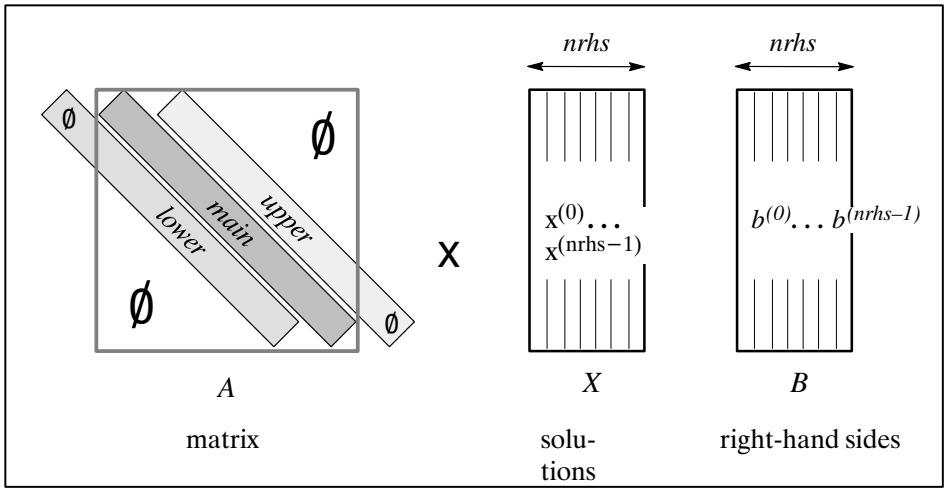

Figure 20. Single tridiagonal system with multiple right-hand sides and solutions.

To solve multiple systems, one specifies the upper, main, and lower arguments with two dimensions: one for the coefficients of the system and one to specify how many systems are represented. The right-hand side (solution) argument is similarly specified in two dimensions (see Figure 21).

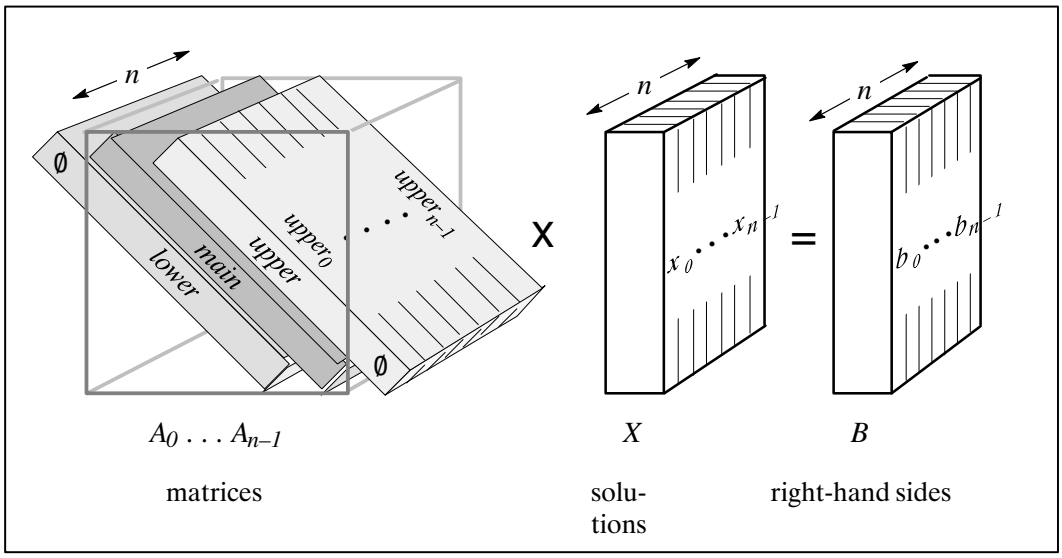

Figure 21. Multiple tridiagonal systems with single right-hand side for each system.

To solve multiple systems each with multiple right-hand sides, one specifies the right-hand-side (solution) argument in three dimensions: one is the length of the

Copyright © 1993 Thinking Machines Corporation

vector, and along this dimension lie the right-hand-side values; one is the number of systems (*n*); and one is the number of right-hand sides (*nrhs*) per system (see Figure 22).

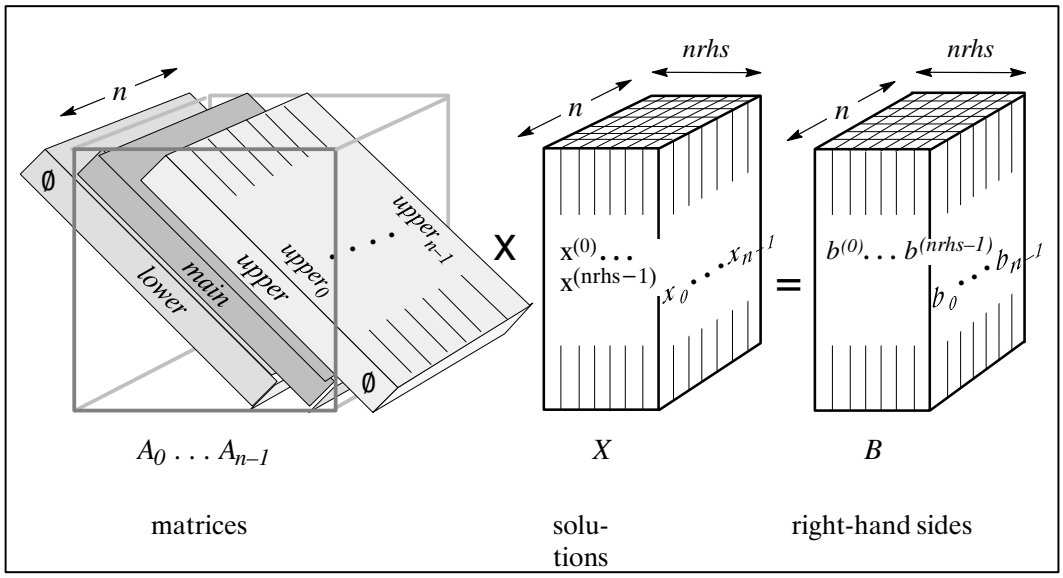

Figure 22. Multiple tridiagonal systems with multiple right-hand sides for each system.

The benefit of using CMSSL routines to solve a single instance of a linear problem lies in the speed gained by exploiting the parallel architecture of the Connection Machine system. Computations on matrices require numerous repetitive calculations along one or both axes. On a serial machine, these must be done one at a time, but on a parallel machine they can be done all at once.

Using CMSSL to solve multiple instances of a linear problem offers similar, but perhaps greater, benefits. For applications that require solving many systems or decomposing many matrices, it is no longer necessary to iterate over the set of systems; the solutions can be computed concurrently.

Copyright © 1993 Thinking Machines Corporation

12.3 Matrix Operations

Basic Linear Algebra Routines for Dense Matrices

The basic linear algebra routines for dense matrices perform the operations listed below. For some operations (inner product, outer product, matrix vector multiplication, vector matrix multiplication, and matrix multiplication), the library includes a family of related routines, each performing a variation on the basic operation. For example, some routines overwrite the supplied destination with the results of the operation; others add the results to values you supply; and some take the transpose, conjugate, or Hermitian of one or more operands.

- *Inner Product Routines*. Compute the inner products of one or more pairs of vectors, or the global inner product over all axes of two arrays. For complex data, you can conjugate the first operand vector.
- *2-Norm Routine*. Computes the 2-norms of one or more real or complex vectors, or the global 2-norm of a real or complex array.
- *Outer Product Routines*. Compute the outer products of one or more pairs of vectors. For complex data, you can conjugate the second operand vector.
- *Matrix Vector and Vector Matrix Multiplication Routines*. Compute one or more matrix vector (or vector matrix) products. For complex data, you can conjugate the matrix.
- *Matrix Multiplication Routines*. Compute one or more matrix products. Variants of the basic routines take the transpose of one or both operand matrices before computing the product; for complex data, you can take the Hermitian of either operand.
- *Infinity Norm Routine*. Estimates the infinity norms of one or more matrices.
- *Matrix Multiplication Routine with External Storage*. Computes a matrix product, where one matrix is too large to fit into core memory and is stored in a file.

November 1993
Copyright © 1993 Thinking Machines Corporation

Basic Linear Algebra Routines for Sparse Matrices

The CMSSL provides routines for basic linear algebra operations on sparse matrices representing structured and unstructured grids. Both elementwise and block sparse matrices are supported. The following operations are provided in each of these categories:

- Sparse matrix x vector
- Vector x sparse matrix
- Sparse matrix x dense matrix
- Dense matrix x sparse matrix

Future versions of the CMSSL will introduce most of the other basic linear algebra operations for arbitrary and grid sparse matrices.

The primary intent of the arbitrary sparse matrix operations is to provide the basic building blocks for more complex sparse applications — for example, a sparse iterative solver, or computation of the eigenvalues of sparse matrices by the Lanczos or the Arnoldi method.

For applications that do not perform explicit sparse linear algebra operations, but want to make use of some communication primitives used by the sparse basic linear algebra functions, the CMSSL provides two utility functions: the sparse gather utility and the sparse scatter utility (described in Section 12.11). These utilities are intended for use in applications such as the solution of partial differential equations on unstructured discretizations, and optimization problems represented by sparse matrices occurring in network flow problems. A communication compiler and a partitioning routine are also provided (see Section 12.11).

Two separate storage representations of arbitrary sparse matrices are supported. These data mappings are referred to as the elementwise sparse matrix mapping and the block sparse matrix mapping. In the elementwise data mapping, the zero data values of the matrix are ignored and the non-zero data values are stored rowwise. In the block sparse mapping, the sparse matrix is stored as a collection of dense block matrices. In its full matrix representation, this block matrix storage scheme is extremely flexible. The dense blocks need not be composed of contiguous rows and columns, and may overlap in any way. One possible application for the block sparse representation is the finite element method. Structured finite element grids lead to a grid block sparse data layout; unstructured grids result in arbitrary block sparse layout.

Copyright © 1993 Thinking Machines Corporation

The grid sparse matrix routines operate on data from grid-based applications. Coefficient matrix elements residing at each grid point *P* are multiplied by vector or matrix elements residing at point *P* and its nearest-neighbor points. The result is placed in product vector or matrix elements residing at point *P*. These routines support multiple instances and block matrices.

The arbitrary and grid sparse matrix functions provide performance optimizations based on the premise that applications will use these sparse functions repeatedly. A marginal setup cost can be incurred before the first call to the sparse functions. The setup cost is then amortized over several calls to the sparse matrix functions.

12.4 Linear Algebraic Equations

Dense Systems of Equations

The CMSSL includes both the *LU* and the *QR* factorization methods for the solution of one or more instances of dense linear algebraic equations:

- *LU Factorization*. These routines use Gaussian elimination (with or without partial pivoting) to factor one or more instances of an $n \times n$ matrix A into a lower triangular matrix L and an upper triangular matrix U, $A=LU$.
- *LU Solution Routines*. These routines use the triangular factors L and U produced by the *LU* factorization routines to produce solutions to the systems $LUX=B$ or $(LU)^{T}X=B$. B may represent one or more right-hand sides for each instance of the systems of equations.
- *QR Factorization*. These routines use Householder transformations (with or without pivoting) to factor one or more instances of an $m \times n$ matrix A, $m \geq n$, into a trapezoidal matrix Q and an upper triangular matrix R, $A=QR$.
- *QR Solution Routines*. These routines use the Q and R factors produced by the *QR* factorization routines to solve one or more instances of the systems of equations $QRX=B$ or $(QR)^{T}X=B$. B may represent one or more right-hand sides for each instance of the systems of equations.
- *Triangular System Solvers*. These routines use the factors produced by the *LU* and *QR* factorization routines to solve triangular systems of equations (trapezoidal systems for Q). Thus, the CMSSL includes routines for solution of one or more instances of triangular systems of equations of the

November 1993
Copyright © 1993 Thinking Machines Corporation

form $LX=B$, $L^TX=B$, $UX=B$, $U^TX=B$, $RX=B$, and $R^TX=B$. B may represent one or more right-hand sides for each instance of the systems of equations.

The CMSSL also contains routines for the solution of one or more instances of trapezoidal systems of equations $QX=B$ or $Q^TX=B$, where again B represents one or more right-hand sides for each instance of the systems of equations.

- *Gauss-Jordan System Solver.* This routine solves (with partial or total pivoting) a system of equations of the form $AX=B$ using a version of Gauss-Jordan elimination. B represents one or more right-hand sides.

- *Matrix Inversion*. This routine inverts a square matrix A using the Gauss-Jordan routine.

- *Utility Routines*. The CMSSL also provides a set of utility routines associated with the factorization routines. For example, there are routines that explicitly compute L, U, and R from the representation used internally in the factorization routines. For QR factorization there are also routines for extracting the diagonal and computing the infinity norm.

- *LU Factorization and Solution with External Storage*. These routines use block Gaussian elimination with partial pivoting to reduce a matrix A to triangular form and solve the system $AX=B$, where A is too large to fit into core memory and is stored in a file.

- *QR Factorization and Solution with External Storage*. These routines use block Householder reflections to perform the factorization $A=QR$ and solve the system $AX=B$, where A is too large to fit into core memory and is stored in a file.

Banded Systems of Equations

Banded linear system solvers solve systems of equations in which the non-zero elements of the coefficient matrix lie in a narrow band around the diagonal. The CMSSL provides routines for solving tridiagonal, pentadiagonal, block tridiagonal, and block pentadiagonal systems of equations. Each routine solves multiple systems of equations, each with one or more right-hand sides, for both real and complex data types. A choice of algorithms is offered.

The multiple-instance capability of the banded system routines in CMSSL is particularly useful in connection with Alternating Direction Methods. You can

Copyright © 1993 Thinking Machines Corporation

specify the axis along which the systems are to be solved. No data reordering or transposition is necessary for the solution of systems along any axis.

The CMSSL includes three types of routines for the solution of banded systems of equations using direct methods:

- *Factorization of Banded Systems.* These routines support five different techniques for the factorization of one or more instances of tridiagonal, block tridiagonal, pentadiagonal, or block pentadiagonal matrices *A*:
  - pipelined Gaussian elimination (with or without partial pivoting)
  - substructuring with pipelined Gaussian elimination
  - substructuring with cyclic reduction
  - substructuring with balanced cyclic reduction
  - substructuring with transpose
- *Solution of Factored Banded Systems*. These routines use the factors produced by the factorization routines to solve one or more instances of systems of equations of the form *LUX=B*, where *L* and *U* are lower and upper (respectively) bidiagonal or block bidiagonal, or lower and upper (respectively) tridiagonal or block tridiagonal matrices, or permutations thereof. *B* represents one or more right-hand sides for each system of equations.
- *Factorization and Solution of Banded Systems*. These routines support the same techniques as above for the solution of one or more instances of tridiagonal, block tridiagonal, pentadiagonal, or block pentadiagonal systems of equations *AX=B*. *B* represents one or more right-hand sides for each system of equations.

Sparse Systems of Equations

The CMSSL includes routines for the solution of sparse systems of equations by iterative techniques. Included are several standard sparse iterative solvers, including Conjugate Gradient (CG), Bi-Conjugate Gradient with Stabilization (BI-CG-STAB), Quasi-Minimum Residual (QMR), and restarted General-Minimum Residual (GMRES).

November 1993
Copyright © 1993 Thinking Machines Corporation

12.5 Eigensystem Analysis

Dense Systems

The CMSSL provides a variety of routines for performing eigensystem analysis of dense systems:

- *Eigensystem Analysis of Dense Hermitian Matrices*. Computes the eigenvalues and eigenvectors of one or more dense real symmetric or complex Hermitian matrices.

- *Eigensystem Analysis using Jacobi Rotations*. Computes the eigenvalues and eigenvectors of one or more dense real symmetric matrices using Jacobi rotations.

- *Selected Eigenvalue and Eigenvector Analysis Using a k-Step Lanczos Method*. Finds selected solutions $\{\lambda, x\}$ to the real standard or generalized eigenvalue problem $Lx = \lambda Bx$. B can be positive semi-definite and is the identity for the standard eigenproblem. The operator L is dense, real, and symmetric with respect to B; that is, $BL = L^{-1}B$. The algorithm used is a k-step Lanczos algorithm with implicit restart.

- *Selected Eigenvalue and Eigenvector Analysis Using a k-Step Arnoldi Method*. Finds selected solutions $\{\lambda, x\}$ to the real standard or generalized eigenvalue problem $Lx = \lambda Bx$. B can be positive semi-definite and is the identity for the standard eigenproblem. The operator L is dense and real. The algorithm used is a k-step Arnoldi algorithm with implicit restart.

- *Generalized Eigensystem Analysis of Dense Symmetric Matrices*. Solves the generalized eigenvalue problem $Ax = \lambda Bx$, where A is dense, real, and symmetric, and B is positive definite.

- *Reduction to Tridiagonal Form and Corresponding Basis Transformation*. These routines reduce one or more dense real symmetric or complex Hermitian matrices to real symmetric tridiagonal form using Householder transformations. After this reduction occurs, for each instance, you can transform the coordinates of an arbitrary set of vectors from the basis of the original Hermitian matrix to that of the tridiagonal matrix, or vice versa.

Copyright © 1993 Thinking Machines Corporation

Tridiagonal Systems

The CMSSL also includes routines for performing eigensystem analysis of real symmetric tridiagonal systems:

- *Eigenvalues of Real Symmetric Tridiagonal Matrices*. Computes the eigenvalues of one or more real symmetric tridiagonal matrices using a parallel bisection algorithm.
- *Eigenvectors of Real Symmetric Tridiagonal Matrices*. Computes the eigenvectors corresponding to a given set of eigenvalues for one or more real symmetric tridiagonal matrices, using an inverse iteration algorithm.

Sparse Systems

The Lanczos and Arnoldi eigensystem analysis routines described above also apply to sparse systems.

12.6 Fourier Transforms

The CMSSL offers routines for the computation of Fourier Transforms by Cooley-Tukey type algorithms on one or more axes of arrays with an arbitrary number of axes. Fast Fourier Transforms (FFTs) have a wide range of scientific and engineering applications including digital filtering of discrete signals, smoothing and decomposition of optical images, correlation and autocorrelation of data series, numerical solution of partial differential equations such as Poisson's equation, and polynomial multiplication.

The CMSSL provides the following FFT routines:

- *Simple Complex-to-Complex FFT.* Performs a complex-to-complex Fast Fourier Transform in the same direction along all axes of a data set.
- *Detailed Complex-to-Complex FFT.* Allows separate specification of the transform direction, scaling factor, and addressing mode along each data axis in a complex-to-complex FFT. Supports multiple instances.

November 1993
Copyright © 1993 Thinking Machines Corporation

- *Detailed Real-to-Complex and Complex-to-Real FFTs*. The real-to-complex FFT computes the Fourier transform of real data; the complex-to-real FFT transforms conjugate symmetric sequences. These operations allow separate specification of the transform direction, scaling factor, and addressing mode along each data axis; they also support multiple instances.

- *Array Conversion Utilities*. These utilities convert real arrays into complex arrays suitable for input to the real-to-complex FFT, and convert complex arrays (supplied in the format produced by the complex-to-real FFT) to real arrays.

12.7 Ordinary Differential Equations

The initial value problem for a system of N coupled first-order ordinary differential equations (ODEs)

$$dy_i(x)/dx = f_i(x,y_1,....,y_N) \quad i=1, \ldots, N \qquad (1)$$

consists of finding the values $y_i(x_1)$ at some value x_1 of the independent variable x, given the values $y_i(x_0)$ of the variables at x_0. The CMSSL provides a routine that solves the initial value problem by integrating explicitly the set of equations (1) using a fifth-order Runge-Kutta-Fehlberg formula. The control of the step size during the integration is automatic. The evaluation of the right-hand side and possibly the scaling array for accuracy control are provided by the user through a reverse communication interface.

12.8 Optimization

The CMSSL provides a routine that solves multidimensional minimization problems using the simplex linear programming method. The goal is to find the minimum of a linear function of multiple independent variables. In the standard formulation, the problem is to minimize the inner product $c^{\mathrm{T}}x$ subject to the condition $Ax = b$, $0 \leq x \leq u$, where A is a matrix, c is a coefficient vector, and $c^{\mathrm{T}}x$ is referred to as the *cost*. The upper bound vector u may be infinity in one or more components.

Copyright © 1993 Thinking Machines Corporation

The simplex routine's reverse communication interface allows you to reinvert (reset the matrix values and restart the routine) when numerical errors accumulate. You can fine-tune the frequency of reinversion and set a tolerance for degeneracy using input arguments.

12.9 Random Number Generation

Two varieties of random number generators (RNG) are included in the CMSSL:

- Fast RNG
- VP RNG

These random number generators use a lagged-Fibonacci algorithm to produce a uniform distribution of random values. This implementation has been subjected to a battery of statistical tests, both on the stream of values within each processor and for cross-processor correlation. The only test that the CMSSL RNGs fail is the Birthday Spacings Test, as predicted by Marsaglia. Despite this failure, these lagged-Fibonacci RNGs are recommended for the most rigorous applications, such as Monte Carlo simulations of lattice gases.

To construct pseudo-random values, the CMSSL random number generators use *state tables*. The Fast RNG allocates one state table per physical Connection Machine node. The VP RNG allocates one state table per array position. The Fast RNG thus consumes substantially less memory than the VP RNG. The VP RNG can produce identical results on differently sized partitions.

Either CMSSL RNG may be reinitialized for reproducible results and checkpointed to guard against the effects of forced interruption.

12.10 Statistical Analysis

The CMSSL statistical analysis routines currently include two histogramming operations. Histograms provide a statistical mechanism for simplifying data. They are generally used in applications that need to display or extract summary information, especially in cases when the raw data sets are too large to fit into the Connection Machine system. Two routines are provided: one that tallies the occurrences of each value in a CM array, and one that counts the occurrences of values within specified value ranges. For particularly large data sets, the range

November 1993
Copyright © 1993 Thinking Machines Corporation

histogram operation facilitates breaking data down into subranges, perhaps as a preliminary step before doing more detailed analysis of interesting areas.

Histograms have many applications in image analysis and computer vision. For example, a technique known as histogram equalization computes a histogram of pixel intensity values in an image and uses it to rescale the original picture.

The CMSSL histogram operations treat the elements of a front-end array as a series of *bins*. In each bin a tally of CM field values or value ranges is stored. The number of histogram bins varies widely with the application, from a dozen tallies on a large process or a few dozen markers on a probability distribution to a few hundred intensity values in an image or a few thousand instruction codes in a performance analysis.

12.11 Communication Functions

The CMSSL includes routines for efficient data motion for nearest-neighbor operations on regular grids, for all-to-all communication on segmented arrays, and for gather and scatter operations on unstructured grids. The library also provides utilities for data distribution for load balancing of communication.

Polyshift

Many scientific applications make extensive use of array shifts in more than one direction and/or dimension in an array geometry. One well-known example is "stencils" used in solving partial differential equations (PDEs) by explicit finite difference methods. Similar communication patterns are encountered in other applications. For example, in quantum chromodynamics one needs to send ($3 \times n$) complex matrices in each direction of a four-dimensional lattice. Multiple array shifts are also useful in many molecular dynamics codes. In the CMSSL, such multiple array shifts are called "polyshifts" (PSHIFTs). They can be recognized in CM Fortran code by a sequence of **CSHIFT** and/or **EOSHIFT** calls in multiple directions of multiple dimensions, with no data dependencies among the arguments and the results of the shifts. There is a potential performance gain in recognizing a polyshift communications pattern, and calling specially developed routines for doing the shifts. In addition, application programs that utilize calls to polyshift routines can benefit from enhanced readability and maintain-

Copyright © 1993 Thinking Machines Corporation

ability. The CMSSL includes a high-level interface for calling polyshift routines from CM Fortran.

All-to-All Broadcast

All-to-all broadcasting is often used to implement data interactions of the type occurring in many so-called *N*-body computations, in which every particle interacts with every other particle. With an array distributed over a number of memory modules, each of which is associated with a parallel processing node, every module must receive the data from every other module. Another example of an application of all-to-all broadcasting is matrix-vector multiplication with the matrix distributed with entire rows per processor, and the vector distributed evenly over the processors. Every processor must gather all the elements of the vector in order to perform the required multiplication.

The CMSSL supports two versions of all-to-all broadcast. One version is intended for applications in which memory requirements are at a premium. For these applications, the all-to-all broadcast can be performed in a stepwise manner in place. The CMSSL also supports such operations for applications in which the all-to-all broadcast can be performed at once.

All-to-All Reduction

In all-to-all reduction, reduction operations such as sum, max, and min are performed concurrently on different data sets, each of which is distributed over all processing nodes; the results of the different reductions are evenly distributed over all nodes. In effect, an all-to-all reduction is the reverse operation of a broadcast, where sum, max, or min replaces the copy operation.

Matrix Transpose

The matrix transpose routine transposes two axes of a multidimensional array. This routine is designed specifically to provide enhanced performance when one of the axes to be exchanged is local (resides within a single processing node or vector unit) and the other is non-local (spans multiple nodes or vector units).

November 1993
Copyright © 1993 Thinking Machines Corporation

Gather and Scatter

The CMSSL includes several gather and scatter utilities:

- *Sparse Gather and Scatter Utilities*. These communication primitives are used by the CMSSL basic linear algebra routines for arbitrary sparse matrices. They are intended for applications that do not do explicit sparse linear algebra operations, but want to make use of some of the primitives commonly used in these operations. The gather utility gathers elements of a vector into an array using pointers supplied by the application; the scatter utility scatters elements of an array to a vector using pointers supplied by the application. Pre-processing is performed by associated setup routines.

- *Enhanced Gather and Scatter Utilities*. These utilities are used in conjunction with the partitioning routine described in the next section. Because the partitioning routine maximizes data locality, the enhanced utilities are significantly faster than the original ones. (The pre-processing time includes the time used to run the partitioning routine, and can be substantial.)

- *Block Gather and Scatter Utilities*. These routines move a block of data elements from a source CM array into a destination CM array. The gather or scatter operation occurs along a single, specified serial axis.

- *Vector Block Gather and Scatter Utilities*. These routines are similar to the block gather and scatter routines, but each index element represents a vector of data items rather than a single data item.

Partitioning of an Unstructured Mesh

The CMSSL provides a routine that allows you to reorder the elements of an unstructured mesh to form discrete partitions. Given an array describing an unstructured mesh, the routine returns a permutation of the mesh elements, the number of resulting partitions, and the number of elements per partition. You can use the permutation to reorder the data you supply to the preprocessor of the enhanced gather and scatter routines (described above). This strategy minimizes the off-vector-unit (or off-processing-node, for machines without vector units) communication required by the gather or scatter operation, since each partition resides within a vector unit (or processing node).

Copyright © 1993 Thinking Machines Corporation

Communication Compiler

The CMSSL communication compiler is a set of routines that compute and use message delivery optimizations for basic data motion and combining operations (get, send, send with overwrite, and send with combining). The communication compiler allows you to compute an optimization (or *trace*) just once, and then use the trace many times in subsequent data motion and combining operations. This feature can yield significant time savings in applications that perform the same communication operation repeatedly. A variety of methods for computing a trace are available.

November 1993
Copyright © 1993 Thinking Machines Corporation