CM Fortran Release Notes

1 Release Highlights

Version 2.1 enhances both the functionality and the performance of CM Fortran. Version 2.1.1 is a bugfix upgrade of V2.1 for the CM-5. The major enhancements are:

- Special compiler optimizations on serial dimensions, which avoid communication and otherwise speed up many local operations.
- Features that give further program control over array layout, including facilities for aliasing arrays of different shapes or layouts, and compiler switches that inhibit axis reordering and local vector-length padding. These features enable a program to get the full benefit of the serial optimizations.
- Two new execution models — nodal and global/local — which move program control to the node level and use CMMD message passing for explicit communication and synchronization. The global/local model provides the means to call node-level subroutines from within a global program.
- Parallel I/O to the Scalable Disk Array and DataVault via the **`READ`** and **`WRITE`** statements, and many other enhancements to CM Fortran I/O (including the keyword **`PHYSICAL`** for the fastest possible parallel I/O).
- Optimizations of **`FORALL`**, of single-precision memory accesses, of the global optimizer **`-O`**, and of operations on dynamically allocated arrays.
- Fortran 90 array pointers (though not pointer assignment) for dynamic management of global storage.
- The 64-bit integer data type and the Fortran 90 **`KIND`** mechanism for managing the "kind," meaning length in bits, of all numeric types.

Copyright © 1994 Thinking Machines Corporation

All 2.1 features are supported on the Connection Machine CM-5 running in global vector-units model (**-vu**). Some of the features are not supported under other execution models, as detailed below in Section 3.

Complete information on these new features appears in the revised CM Fortran documentation set for Version 2.1 (see Section 4). For information on using message passing in CM Fortran programs, see the CMMD documentation set.

2 CM Platforms and Execution Models

Version 2.1 supports all CM platforms and offers two new execution models that permit node-local programming with message passing. The supported execution models are:

- CM-5 release of the compiler
  - global vector-units model (**-vu**)
  - global SPARC-nodes model (**-sparc**)
  - nodal CM Fortran (**-vu -node**)
  - global/local CM Fortran (**-vu ... -local**)
  - CM Fortran simulator (**-cmsim**)
- CM-2/200 release of the compiler
  - slicewise model (**-slicewise**)
  - CM Fortran simulator (**-cmsim**)

Version 2.1 does not support the Paris execution model on the CM-2 or CM–200 (**-paris**).

Copyright © 1994 Thinking Machines Corporation

3 New Features by Execution Model

This section lists all the new features and major compiler optimizations provided in Version 2.1. All these items are available on the CM-5 global vector-units model (**-vu**). Some features do not support all execution models, as noted below.

NOTE: The nodal and global/local models are variants of the global vector-units model and share all its features except I/O. For I/O from nodal CM Fortran, use CMMD I/O. Global/local programs (using CM Fortran Version 2.1 and CMMD Versions 3.1 or 3.2) require that all I/O be done from a global program unit, using any of the CM Fortran I/O facilities; no I/O is permitted from local subroutines.

Table 2. Supported features by execution model.

Key: yes = feature is supported
– = feature has no effect
no = feature is not accepted (program fails)

Features	CM-5		CM-2	Sun
	–vu	**–sparc**	**–slice**	**–cmsim**
▪ optimizations on serial dimensions - no-motion detection - nested subgrid loop, perhaps flattened - context handling in serial sections - simple dependence analysis - contiguous sections passed in place - gathers/scatters with **FORALL** - spreads with **FORALL** or **SPREAD**	yes	yes	–	–
▪ user control over array layout				
- array aliasing package (14 procedures)	yes	yes	yes	–
- controllable axis sequence, **-[no]axis**	yes	yes	no	no
- controllable local padding, **-[no]pad**	yes	–	no	no

Copyright © 1994 Thinking Machines Corporation

Table 1. Supported features by execution model *(cont'd)***.**

Features	CM-5		CM-2	Sun
	−vu	−sparc	−slice	−cmsim
▪ node-level programming with message passing - the nodal execution model - the global/local execution model - the CMGL library procedures	yes	no	no	no
▪ optimizations				
- global optimizer improvements	yes	yes	yes	yes
- geometry lookups in subprograms	yes	yes	yes	–
- run-time check of dynamic array geometries	yes	yes	yes	–
- compile-time check of dynamic array shapes	yes	yes	yes	–
- **FORALL** scans parallelized	yes	yes	–	–
- single-precision **REAL** stores	yes	–	–	–
- single-precision **COMPLEX** loads/stores	yes	–	–	–
▪ I/O enhancements in CM Fortran language				
- parallel **READ/WRITE** to SDA	yes	yes	no	–
- **READ/WRITE** character data to SDA	yes	yes	no	yes
- parallel **READ/WRITE** to DataVault	yes	yes	yes	–
- **READ/WRITE** character data to DataVault	yes	yes	no	–
- **PHYSICAL** keyword for **OPEN**	yes	yes	yes	–
- 4 new specifiers to **OPEN** and **INQUIRE**	yes	yes	yes	yes
- new **NAMELIST** syntax	yes	yes	yes	yes
- **REWIND/BACKSPACE** on redirected output	yes	yes	yes	yes

Copyright © 1994 Thinking Machines Corporation

Table 1. Supported features by execution model *(cont'd)***.**

Features	CM-5		CM-2	Sun
	−vu	**−sparc**	**−slice**	**−cmsim**
▪ I/O enhancements in CM Fortran Utility Library				
- SDA support via utility library I/O	yes	yes	no	no
- utility **CMF_FILE_OPEN_READONLY**	yes	yes	yes	no
▪ array pointers	yes	yes	yes	yes
- **POINTER** attribute				
- **POINTER** statement				
- enhanced **ALLOCATE** and **DEALLOCATE**				
- **ASSOCIATED** intrinsic function				
▪ the Fortran 90 **KIND** mechanism	yes	yes	yes	yes
- **KIND** keyword				
- kind type parameters (except **_DOUBLE_INT_**)				
- syntax for typed constants				
- **KIND** intrinsic function				
▪ 64-bit integers	yes	no	no	no
- **INTEGER*8** data type				
- kind type parameter **_DOUBLE_INT_**				
- utility **CMF_RANDOM_LONG_S_INTEGER**				
- utility **CMF_FILE_LLSEEK**				
- utility **CMF_FILE_LTRUNCATE**				
- arg parameter **CMF_LONG_S_INTEGER**				
- optional **KIND** arg to **INT** and **NINT**				
- **INTEGER*8** send-address arrays				

Copyright © 1994 Thinking Machines Corporation

4 Current Documentation

There is a new CM Fortran documentation set for Version 2.1. Manuals are available in hardcopy, on-line in CMview (in collection CM5–2 or later), and on-line in PostScript and ASCII formats.

4.1 Manual Titles

The new V2.1 documentation set consists of:

- *CM Fortran Language Reference Manual*
- *CM Fortran Libraries Reference Manual*
- *CM Fortran User's Guide*
- *CM Fortran Programming Guide*
- *CM-5 CM Fortran Performance Guide*
- *CM Fortran Array Operations Quick Reference Guide*

Also provided are:

- *Getting Started in CM Fortran*, January 1993
- *CM Fortran for the CM-5 Release Notes* [this document]
- C*M Fortran for the CM-200 Release Notes*

4.2 Finding the Manuals

All the V2.1 manuals are available in hardcopy.

The CMview document collection CM5–2 provides:

- On-line viewing and search of all the documents except release notes and *Quick Reference Guide*.
- PostScript versions of all the documents (except *Quick Reference Guide*), which are intended for printing. These PostScript documents are typically installed on CM-5 systems under **`/usr/doc/cmfortran`**.

ASCII versions of the release notes and the manuals can be accessed through Prism.

Copyright © 1994 Thinking Machines Corporation

4.3 On−Line man Pages

This release provides up-to-date on-line man pages for:

- **cmf** command
- all intrinsic functions
- all procedures in the CM Fortran libraries:
  - utility library
  - global/local library
  - cmf77 library

View the procedure man pages with the command **man**, followed by the procedure name in uppercase. (The utility procedures can be specified in lower or mixed case, but the intrinsics and cmf77 routines must be specified in uppercase to avoid name conflicts with Sun man pages.)

4.4 Detailed Release Notes

A document called *CM Fortran Release Notes: Detailed*, V2.1, is included in PostScript under **/usr/doc/cmfortran**. This manual gives full information on all new features and compiler optimizations in Version 2.1.

The detailed release notes are intended only as a convenience for users whose V2.1 compiler is installed before the documentation. Once the V2.1 documentation set is available (in hardcopy, CMview, and/or PostScript), the detailed release notes are redundant and can be discarded.

4.5 Doc Corrections

Fortran READ/WRITE and the SO Utilities

This note corrects *CSG Technical Bulletin CMF 2.1-Beta-002*, "serial−order−IO=direct−access," June 28, 1993.

That bulletin states that the CM Fortran Utility Library I/O procedures **CMF_CM_ARRAY_TO/FROM_FILE_SO** are compatible with Fortran **READ/WRITE** with **ACCESS=DIRECT**. That is not correct. Mixing these two I/O mechanisms sometimes causes an end-of-file error and is not supported. CM For-

Copyright © 1994 Thinking Machines Corporation

tran **READ/WRITE** are fully compatible with Fortran 77 **READ/WRITE**, as well as now supporting parallel I/O.

CMF_FILE_FDOPEN

The order of arguments to the utility library procedure **CMF_FILE_FDOPEN** was documented incorrectly in the *CM Fortran Utility Library Reference Manual*, Version 2.0 Beta. The correct syntax is:

```
CALL CMF_FILE_FDOPEN ( UNIT, CMFD_FD, IOSTAT )
```


Copyright © 1994 Thinking Machines Corporation

5 Porting Existing Programs

Codes developed under previous releases must be recompiled and relinked to execute under Version 2.1.

Correct programs require no code changes for V2.1. However, two changes in the implementation require recoding of unsupported features that previously worked. Also, the library procedure **FLUSH** may cause a linking problem in programs compiled for the nodal execution model.

5.1 Changing Array Type or Shape Across Program Boundaries

In previous releases, the geometry associated with a CM array argument was determined upon entry to a subroutine based on the declaration in the subroutine. In the current release, the geometry of a CM array argument is determined from the descriptor of the actual argument (at the caller level).

A side effect (unsupported) of the previous behavior was that it allowed limited forms of equivalencing to occur at program boundaries. The new behavior prevents this.

For example, the following program fails because the geometry used for **B** is the geometry stored in **A**'s descriptor, thus violating the CM Fortran restriction against changing array shape across program boundaries:

```
      REAL A(10,100)
CMF$  LAYOUT A(:SERIAL,:NEWS)
      CALL FOO(A)
      ...
      SUBROUTINE FOO(B)
      REAL B(2,5,100)     ! Error: shape mismatch
CMF$  LAYOUT B(:SERIAL,:SERIAL,:NEWS)
      ...
```

Similarly, the following program fails because of the attempt to change array **C**'s type and shape in the subroutine:

```
      COMPLEX C(8,128)
CMF$  LAYOUT C(:BLOCK=8 :PROCS=1,:BLOCK=1 :PROCS=128)
      CALL BAR(C)
      ...
      SUBROUTINE BAR(D)
```


Copyright © 1994 Thinking Machines Corporation

```
      REAL D(16,128)      ! Error: type/shape mismatch
CMF$  LAYOUT D(:BLOCK=16:PROCS=1,:BLOCK=1 :PROCS=128)
```

5.2 Passing Array Sections without Interface Blocks

A change in the way the compiler passes array sections may require code changes to avoid errors in programs that do not use interface blocks.

There are now situations where an argument array section is passed in place (such as when a contiguous section is defined with a triplet subscript on a serial axis), whereas previously it was copied to a canonical temporary. This change will cause your program to fail if the called procedure is defined to expect a canonical argument.

For example, given an array laid out **A(:SERIAL,:NEWS)**, the default switch **-axisreorder**, and no interface block, consider the call

```
CALL FOO( A(3:5,:) )
```

Previously, the argument would have been passed as a **(:NEWS,:NEWS)** temporary, so **FOO** might be written with no **LAYOUT** directive or with one that specifies the dummy argument as the default **(:NEWS,:NEWS)** layout. Such a program will fail now that the section is passed in place as **(:SERIAL,:NEWS)**.

See the *CM-5 CM Fortran Performance Guide*, Version 2.1, for information on contiguous array sections; notice that changing the setting of the switch **-[no]axisreorder** can affect contiguity and thus determine whether a section is passed in place.

Programs that use interface blocks will not be affected by this change.

5.3 Note on Back-Compatibility of Utility I/O

The CMFS I/O routines underlying the utility library procedures **CMF_CM_ARRAY_TO/FROM_FILE** no longer pad array data to 16-byte "CM word" boundaries. However, to preserve compatibility with files written under previous CM Fortran releases, the CM Fortran procedures add the padding on writes and remove it on reads. No code changes are needed in CM Fortran programs.

Copyright © 1994 Thinking Machines Corporation

REMINDER: Always read a CM file with the same mechanism that was used to write it. That is, read with **READ** if written with **WRITE**, read with **CMF_...SO** if written with the **_SO** routine, and so on.

5.4 Avoiding Calls to FLUSH in Nodal Programs

Future releases of CM Fortran will not support the cmf77 library procedure **FLUSH**. We recommend that you remove calls to **FLUSH** from CM Fortran programs. Removal has no effect, since CM Fortran now flushes I/O buffers automatically after every write operation (not just at program exit).

Nodal programs that call **FLUSH** and also include Fortran 77 modules may encounter a linking error, since the procedure is also defined in Sun libraries. Removing **FLUSH** from the CM Fortran code avoids the error.

Copyright © 1994 Thinking Machines Corporation

6 Restrictions

6.1 Restrictions Imposed

- The **INTEGER*8** data type and related features are supported only under the vector-units execution model (**-vu**) on the Connection Machine CM-5. Under this model, **INTEGER*8** data can be used on both the partition manager and the processing elements.
- CM Fortran 2.1 for the CM-5 does not support the Connection Machine models CM-200 and CM-2. A separate release of Version 2.1 supports those platforms, but only the slicewise execution model (**-slice**).
- CMSSL 3.1 does not support the CM Fortran switches **-nopadding** and **-noaxisreorder**. Because CM Fortran requires that all program units in a program be compiled with the same setting of these switches, no program unit can call CMSSL routines if either of these switches is used.
- Prism 1.2 and 2.0 do not support profiling CM Fortran global/local programs. Do not use **-cmprofile** together with **-local**.
- CMMD 3.1 and 3.2 do not support any I/O from a local subroutine in a CM Fortran global/local program. Perform all I/O from a global program unit using CM Fortran I/O facilities.

Feature-specific restrictions are noted in the documentation of the feature. See also the **.bugupdate** file, accessible on-line through Prism.

Copyright © 1994 Thinking Machines Corporation

6.2 Restrictions Removed

- Previous CM Fortran versions required that all **COMMON** blocks containing CM arrays be declared in the main program unit, regardless of where in the program they were used. This restriction is removed in V2.1.

- Previous CM Fortran versions set a limit of 20 on the number of **INCLUDE** files that could be used in a source file. The compiler now accepts up to 252 **INCLUDE** files per source file, nested to a maximum depth of 19.

  The total refers to distinct files referenced in all the **INCLUDE** lines in all the compilation units in a source file. If, in the same file, **SUB1** includes **inc1.fcm**, **inc2.fcm**, and **inc3.fcm**, and **SUB2** includes **inc3.fcm** and **inc4.fcm**, the total charged against the 252 is 4.

- Previous versions limited Fortran unit numbers to the range 0:100. The range has been increased to 0:2999, for both the CM Fortran **OPEN** statement and the utility procedure **CMF_FILE_OPEN**.

Copyright © 1994 Thinking Machines Corporation

7 Status of FORALL Implementation

Many of the previous performance-related restrictions on the use of **FORALL** have been removed in Version 2.1.

7.1 Operations Now Optimized

Gathers/Scatters on Serial Dimensions

When **FORALL** expresses a gather or scatter operation on a serial axis, the compiler can determine that the operation is entirely local and uses the fast indirect addressing hardware. Examples of optimized indirect addressing:

```
      INTEGER SOURCE(NS,NP), DEST(NP), INDEX(NP)
CMF$  LAYOUT SOURCE(:SERIAL,), DEST(), INDEX()
      ...
      FORALL (I=1:NP) DEST(I)=SOURCE((I),I)
```

and,

```
      INTEGER, ARRAY(NS,NP) :: SOURCE, DEST, INDEX
CMF$  LAYOUT SOURCE(:SERIAL,),DEST(:SERIAL,)
CMF$  LAYOUT INDEX(:SERIAL,)
      ...
      FORALL (I=1:NS,J=1:NP)
     &   DEST(I,J)=SOURCE(INDEX(I,J),J)
```

This form of **FORALL** on serial dimensions gives performance comparable to the utility library procedures **CMF_AREF/ASET_1D**.

Spread Operations on Serial Dimensions

When **FORALL** expresses a spread operation on a serial axis, the compiler can determine that the operation is entirely local and generates local memory accesses instead of communication routines. Example of an optimized spread:

```
      REAL A(10,10), B(10)
CMF$  LAYOUT A(:SERIAL,:NEWS), B(:NEWS)
      ...
      FORALL(I=1:10) A(I,:) = B(:)
```


Copyright © 1994 Thinking Machines Corporation

Scan Operations

FORALL now generates run-time scan routines, including conditional and segmented scans and scans of scans, rather than generating the less efficient reduction-of-spread expressions. **FORALL** now gives performance comparable to the utility library procedures **CMF_SCAN_***combiner*.

Examples of optimized scans:

```
FORALL(I=1:N) A(I) = SUM(B(1:I))

FORALL(I=1:N,J=1:M) A(I,J) = SUM(B(I,1:J))

FORALL (I=1:N) A(I) =
&    SUM(B(MAXVAL([1:I],SEGMENT(1:I)):I))

FORALL(I=1:N,J=1:M) A(I,J) = SUM(B(1:I,1:J))
```

The scan capabilities of **FORALL** are now largely, but not entirely, the same as those of the utility library procedures **CMF_SCAN_***combiner*. The comparative combiners are:

CMF_SCAN_*combiner*	**FORALL** expression
ADD	**SUM**
MAX	**MAXVAL**
MIN	**MINVAL**
mult scan (not available)	**PRODUCT**
LOGICAL IOR	**ANY**
LOGICAL IAND	**ALL**
LOGICAL IEOR	**MOD(COUNT(),2)==1**
INTEGER IOR	(not available)
INTEGER IEOR	(not available)
INTEGER IAND	(not available)

To do a copy scan, you can use a vector-valued subscript instead of the sum of a triplet expression. The triplet is `MAXVAL([1:I],SEGMENT(1:I))` as the lower bound and I is the upper bound.

Copyright © 1994 Thinking Machines Corporation

7.2 Expressions Now Parallelized

The **FORALL** statement now accepts the following expressions and generates parallel expressions.

- References to all intrinsic functions and a few specific function calls that are listed below under "restrictions."

  Previous versions of **FORALL** always serialized the transformational functions **CSHIFT**, **EOSHIFT**, **DIAGONAL**, **DOTPRODUCT**, **FIRSTLOC**, **LASTLOC**, **MATMUL**, **MAXLOC**, **MINLOC**, **PACK**, **PROJECT**, **REPLICATE**, **RESHAPE**, **SPREAD**, **TRANSPOSE**, **UNPACK**. and the inquiry functions **DSIZE**, **DSHAPE**, **DLBOUND**, **DUBOUND**, and **RANK**. As of Version 2.0, **FORALL** accepts these functions, except in the cases noted below.

- Use of a **FORALL** index variable in the following ways:
  - in a triplet subscript, such as `A(1:I)` or `B(I:I+5:2)`
  - in an array constructor, such as `[I]` or `[1:I]`
  - as an argument to a statement function, such as `FOO(I)`

7.3 Temporary Restrictions

FORALL has several temporary inefficiencies, where it either executes serially or fails to generate the most efficient parallel instructions.

- Temporarily, **FORALL** executes serially if it references:
  - the intrinsic function **RESHAPE** with a **PAD** argument, with triplet **SOURCE**, or with **MOLD** or **ORDER** arguments not specified with literal constants

- Temporarily, **FORALL** executes serially if an index variable is used in any of the following expressions:
  - front-end array expression
  - unaligned binary triplet expression, such as `B(0:I) + B(I:2*I)`
  - multiple-term array constructor, such as `[B(I),C(I),D(I)]`

Copyright © 1994 Thinking Machines Corporation

- multiple-dimension expression to **MAXLOC** or **MINLOC**, such as `MAXLOC(B(I,:,:))`
- **REPLICATE** along triplet axis, such as `REPLICATE(B(1:I),DIM=1,NCOPIES=8)`

- Temporarily, **FORALL** does not generate the most efficient parallel instructions for array transfers between the parallel processors and the control processor. A simple CM-to-FE transfer operation is expressed as:

```
FORALL (I=1:N) FE(I) = CM(I)
```

  In the current release, this statement generates a serial **DO** loop with a **read-to-processor** or **write-from-processor**. It is better to use the utility procedures **CMF_FE_ARRAY_TO/FROM_CM**.

7.4 Permanent Restrictions

The following cause **FORALL** to execute serially:

- Reference to a character variable, such as `STRING(I)`.
- Reference to an external function.
- "Too many" triplet axes, such as `A(1:I,1:J,1:K,1:L)`. The operation executes serially if the number of vector axes (those with colons) plus the number of distinct **FORALL** indices referenced is above a certain threshold. The threshold in the current release is 7. The expression above references 4 vector axes and 4 indices, for a total of 8, and thus executes serially.
- **CSHIFT** or **EOSHIFT** of a triplet axis using a **FORALL** index, such as `CSHIFT( B(1:I),1,1 )`.
- A nonconstant **DIM** argument, that is, a dimension not known at compile time, to any reduction function, or to the transformational functions **CSHIFT** or **EOSHIFT**, **FIRSTLOC** or **LASTLOC**, **PROJECT**, **REPLICATE**, **SPREAD**, or to the inquiry functions **DUBOUND**, **DLBOUND**, or **DSIZE**.

Another restriction is that **FORALL** does not support assumed-size arrays (those declared with * as the last axis extent) or the type **CHARACTER**.

Copyright © 1994 Thinking Machines Corporation

8 Notice of Possible Future Changes

8.1 Phasing Out Linking with Sun f77 Libraries

CM Fortran automatically links with Sun libraries if they are installed on the system. This behavior will be discontinued in a later version of CM Fortran.

Even in the current version, you can get a run–time error for linking with Sun's dynamically bound libraries. When a program that was linked with these libraries does not find them at run time, the following error is signalled:

```
ld.so: libF77.so.1: not found
```

You can prevent this problem by linking statically by means of the **-Bstatic** switch, which **cmf** passes on to the linker:

```
% cmf myfile.fcm -Bstatic
```

8.2 Cmf77 Procedure FLUSH Deprecated

Future releases of CM Fortran will not support the cmf77 library procedure **FLUSH**. We recommend that you remove calls to **FLUSH** from CM Fortran programs. Removal has no effect, since CM Fortran now flushes I/O buffers automatically after every write operation (not just at program exit).

Nodal programs that call **FLUSH** and also include Fortran 77 modules may encounter a linking error, since the procedure is also defined in Sun libraries. Removing **FLUSH** from the CM Fortran code avoids the error.

8.3 Axis Ordering in Memory

The current default axis ordering, **-axisreorder**, exists in the compiler for historical reasons. The **-noaxisreorder** switch produces the standard Fortran left-to-right ordering. In the future, the left-to-right ordering may become the default. We encourage its use.

NOTE: -axisreorder affects only CM arrays. Front-end array axes are always stored in left-to-right order, regardless of the setting of this switch.

Copyright © 1994 Thinking Machines Corporation

8.4 Vector-Length Padding

Similarly, the current default padding of the (non-serial) subgrid to a multiple of vector length also exists for historical reasons. (The compiler did not originally generate clean-up code for loops in which the number of times the loop is executed is not a multiple of the vector length.) In the future, **-nopadding** may replace **-padding** as the default. We encourage its use.

8.5 Compatibility

The previous defaults for **-[no]axisreorder** and **-[no]padding** have been retained in this release for back-compatibility. If the defaults change, or if you begin to switch over to the nondefault options, consider compatibility issues like the following:

- Suppose you have a library compiled under the current defaults (**-axisreorder** and **-padding**). In the future, when the defaults change, you may try to call this library from a newly compiled procedure (with **-noaxisreorder** and **-nopadding**). The program will fail because all program units in a program must be compiled with the same setting of these switches.
- Suppose you have an array **A** laid out **(:SERIAL,:NEWS)** and you pass **A(1,:)** to a subroutine. With the current default **-axisreorder**, this section is passed in place. In the future, with the **-noaxisreorder** default, the section would first be sent to a temporary location, thus slowing down the program.

Copyright © 1994 Thinking Machines Corporation