CMAX 2.0
Release Notes

1 About Version 2.0

1.1 Release Highlights

The major enhancements over the previous release are:

- Greater robustness and reliability
- New conversion capabilities:
  - reuse of introduced arrays to save memory
  - array promotion to permit conversion
  - statement splitting to permit conversion
- Support for partial or incremental program conversion in "firewalls" mode, which produces output that is object-code-compatible with the remainder of the original Fortran 77 program
- More convenient linking of the CMAX library, using **`-lcmax`**, beginning with CM Fortran 2.2

1.2 Moving from Version 1.0

Please remove all packages created under Version 1.0 before beginning to use 2.0.

Copyright © 1994 Thinking Machines Corporation

1.3 Information

The current documentation for this release is:

- *CMAX 2.0 Release Notes* — hardcopy and PostScript
  `cmax-2.0.releasenotes` — ASCII

  This document describes all features added for this release.

- **`cmax-2.0.bugupdate`**

  This ASCII file reports all bugs known at release time.

- **`cmax`** man page

  The on-line man page for the **`cmax`** command is accessible via the **`man`** command on CM-5 and standalone systems or **`cmman`** on CM-200 and CM-2 systems.

- *Using the CMAX Converter,* Version 2.0, May 1994

  This manual is available in hardcopy or on line in PostScript format; it will appear shortly in CMview.

The pathname leading to the on-line documents is typically

- **`/usr/doc/`** — on CM-5 and standalone Sun-4
- **`/usr/cm/doc/`** — on CM-200 and CM-2

See your system administrator for the location if these files have been moved.

Copyright © 1994 Thinking Machines Corporation

1.4 On-Line Examples and Makefiles

The pathname leading to the examples directory is typically

- **/usr/examples/cmax/** on CM-5 and standalone Sun-4
- **/usr/cm/examples/cmax/** on CM-200 and CM-2

Directories beneath **examples/cmax/** contain:

- **cm2/** — an example and sample Makefile for using CMAX and CM Fortran with the CM-2 slicewise model.
- **cm5/global/** — an example and sample Makefile for using CMAX and CM Fortran with the CM-5 global vector-units execution model.
- **cm5/nodal/** — an example and sample Makefile for using CMAX and CM Fortran with the CM-5 nodal execution model.
- **cm5/firewalls/** — an example and sample Makefile for using CMAX "firewalls" and CM Fortran with the CM-5 global vector-units execution model. CMAX "firewalls" may be used with other programming models, and with the CM-2, by adjusting the Makefile appropriately. Use the other sample Makefiles as guides.
- **emacs**/ — an updated version of the EMACS tools.
- **cmax-library/** — an updated version of the portable source code for the CMAX library.

See your system administrator for the location if these files have been moved.

Copyright © 1994 Thinking Machines Corporation

2 New Conversion Capabilities

In Version 2.0, CMAX can translate certain kinds of DO loops that it could not translate before, and generates more space-efficient CM Fortran code.

2.1 Reuse of Introduced Arrays

CMAX 1.0 did not attempt to reuse the arrays it introduced into the program (such as promoted scalar arrays) in all the circumstances possible. This was a problem since certain styles of programming can lead to many such temporary arrays, and in many applications memory is at a premium.

CMAX 2.0 does reuse promoted-scalar arrays, arrays used for computing send addresses, and other introduced arrays, thus decreasing memory requirements of converted programs.

For example, this set of loops:

```
DO I = 1,100
   A(I) = I
END DO
DO I = 1,100
   X = A(I)
   A(I) = X + X
END DO
DO I = 1,100
   Y = A(I)
   A(I) = Y + Y
END DO
DO I = 1,100
   Z = A(I)
   A(I) = Z + Z
END DO
```


Copyright © 1994 Thinking Machines Corporation

was converted by CMAX 1.0 into:

```
REAL  X100(100)
REAL  Y100(100)
REAL  Z100(100)
...
FORALL (I = 1:100) A(I) = I
X100 = A
A = X100 + X100
Y100 = A
A = Y100 + Y100
Z100 = A
A = Z100 + Z100
```

requiring three introduced arrays, **X100**, **Y100**, and **Z100**.

In CMAX 2.0 that set of loops converts to code that introduces only array **X100** and reuses it:

```
REAL  X100(100)
...
FORALL (I = 1:100) A(I) = I
X100 = A
A = X100 + X100
X100 = A
A = X100 + X100
X100 = A
A = X100 + X100
```


Copyright © 1994 Thinking Machines Corporation

2.2 Array Promotion

While CMAX 1.0 promoted scalars to arrays in cases where doing so made parallel execution possible, CMAX 2.0 goes further. It promotes arrays to arrays of higher rank where parallelism will be increased.

For example, this code uses 2D array **A** together with 1D array **B**:

```
DO J = 1,30
    DO I = 1,30
        B(I) = A(I,J) ** 1.2
        B(I) = B(I) + SQRT(B(I))
        A(I,J) = B(I) * 3.0
    END DO
END DO
```

CMAX 1.0 parallelized this loop along only one dimension of **A**:

```
DO J = 1 , 30
   B = A(:,J) ** 1.2
   B = B + SQRT(B)
   A(:,J) = B * 3.0
ENDDO
```

CMAX 2.0 promotes array **B** to 2D array **B100** and generates code that allows parallel execution along both dimensions.

```
REAL  B100(30,30)
...
B100 = A ** 1.2
B100 = B100 + SQRT(B100)
A = B100 * 3.0
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

A limitation of array promotion is that it occurs only when there is a loop in the nest that goes along the array dimension. Thus, array **zz** in this code is not promoted:

```
DO K = 1,30
  DO J = 1,30
    DO I = 1,30
      ZZ(1) = A(I,J,K) * 1.0
      ZZ(2) = A(I,J,K) * 2.0
      ZZ(3) = A(I,J,K) * 3.0
      A(I,J,K) = ZZ(1) + ZZ(2) + ZZ(3)
    END DO
  END DO
END DO
```

There are two workarounds for this limitation. In some cases, as in the example above, you can rewrite the code by introducing a new loop nest:

```
DO K = 1,30
  DO J = 1,30
    DO I = 1,30
      DO IDX = 1,3
        ZZ(IDX) = A(I,J,K) * IDX
      END DO
      A(I,J,K) = ZZ(1) + ZZ(2) + ZZ(3)
    END DO
  END DO
END DO
```


Copyright © 1994 Thinking Machines Corporation

In other cases, the array can be dispensed with and scalars used instead. The scalars will be promoted in the usual way:

```
DO K = 1,30
  DO J = 1,30
    DO I = 1,30
      ZZ1 = A(I,J,K) * 1.0
      ZZ2 = A(I,J,K) * 2.0
      ZZ3 = A(I,J,K) * 3.0
      A(I,J,K) = ZZ1 + ZZ2 + ZZ3
    END DO
  END DO
END DO
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

2.3 Statement Splitting

CMAX 1.0 did not convert reductions of expressions that could not be represented in simple array syntax; they remained as serial **DO** loops in the output. CMAX 2.0 can split assignment statements that, inside **DO** loops, comprise such reductions, generating a separate **FORALL** statement and reduction assignment.

For example, CMAX 1.0 could not parallelize this loop because the right-hand side cannot be expressed in array syntax:

```
REAL A
REAL X(N),X2(N)
...
DO I = 1,N-3
    A = A + ABS(I + X(I+1) * SQRT(X2(I+2)))
ENDDO
```

CMAX 2.0, however, converts it to two statements:

```
REAL A
REAL X(N),X2(N)
REAL TEMP(N)            ! temporary CM array
...
FORALL (I = 1:N - 3)
&   TEMP(I+1) = ABS(I + X(I+1) * SQRT(X2(I + 2)))
A = A + SUM(TEMP(2:N - 2))
```

Notice that this enhancement makes it generally unnecessary to recode complicated expressions manually to reveal reduction idioms to CMAX (as recommended in the Version 1.0 CMAX manual).

Copyright © 1994 Thinking Machines Corporation

3 Partial Program Conversion with "Firewalls"

CMAX 2.0 provides a more convenient way to convert programs partially or incrementally. In "firewalls" mode, CMAX manages the interfaces between the converted and unconverted portions of the program.

In Version 1.0 partial conversion was limited to a single subtree at a time, and the user had to manage the interface between the converted and unconverted program units — particularly with regard to array homes. In some cases, the converted output could not be linked with the remainder of the original program into a single executable because of array home mismatches across the program boundaries. More often, some computation was not vectorized because the arrays it used were constrained to a front-end home elsewhere in the program.

In Version 2.0, the CMAX user can identify one or more kernels of computationally intensive subroutines on which to focus the conversion effort. In "firewalls" mode, the kernel is converted into CM Fortran while the part outside the kernel remains in Fortran 77. All arrays, including arrays in **COMMON**, that are used in a parallel way within the kernel, are copied to CM arrays, and, if modified by the kernel, are copied back to front-end arrays on exit. By confining the conversion effort to the kernel, many problems that might restrict array homes to the front end — which often occur in parts of the program that are not computationally intensive — can be avoided.

3.1 Setting Up Firewalls

The user specifies the boundary at which "firewalls" are erected with the **-InterfaceRoutines=** option, followed by one or more program unit names.

The output generated from an identified interface routine **FOO** is two separate routines:

- **FOO_V1** is the original routine converted to CM Fortran
- **FOO** becomes a new interface routine that does the following:
  - creates CM arrays corresponding to all the arrays used in array operations in any of its callees

Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

- transfers data from each front-end (Fortran 77) array to the corresponding CM array
- calls the converted routine **FOO_V1**
- transfers data from CM arrays back to the corresponding front-end arrays if the data has been modified.

For example, Figure 1 shows a simple routine that has been identified as a firewall interface, along with the two routines CMAX generates.

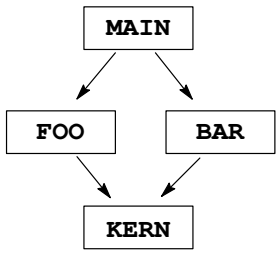
```
    % cmax package-name -T -InterfaceRoutines=FOO


Input in foo.f

      SUBROUTINE FOO(X)
      REAL X(100)
      DO I = 1,100
         X(I) = X(I) + I
      END DO
      END


Output in foo.fcm

      SUBROUTINE FOO(X)     ! generated interface routine
      REAL X(100)
CMF$  LAYOUT X(:SERIAL)     ! its FE array argument
      REAL X0(100)
CMF$  LAYOUT X0(:NEWS)      ! a local CM array
      CALL CMF_FE_ARRAY_TO_CM(X0,X)   ! FE-CM data transfer
      CALL FOO_V1(X)        ! call converted routine below
      CALL CMF_FE_ARRAY_FROM_CM(X,X0) ! CM-FE data transfer
      END

      SUBROUTINE FOO_V1(X)  ! the converted routine
      REAL X(100)
CMF$  LAYOUT X(:NEWS)       ! its CM array argument
      FORALL (I = 1:100) X(I) = X(I) + I
      END
```

Figure 1. Conversion of a specified interface routine.

Copyright © 1994 Thinking Machines Corporation

The package used in "firewalls" conversion should contain the whole program, ideally with each routine in a separate file. The **fsplit** program, available on most Sun systems under **/usr/lang/**, can be used to split files containing multiple subprograms into files containing one subprogram each.

Also, the argument to **-Interface** can be either a program unit name or a list of names. It does not matter whether the subtrees under the interface routines are separate or interdependent. For example, imagine that program **MAIN** calls subroutines **FOO** and **BAR**, and both call the computational kernel **KERN**.



Here, it might make sense to erect the firewall just below **MAIN**:

```
% cmax mypack -Add=main.f foo.f bar.f kern.f
% cmax mypack -T -InterfaceRoutines=foo bar
```

It is also advisable to use the option **-OutputExtension=**, for reasons that will become clear in Section 3.2.

Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

3.2 A More Elaborate Example

This section contains a more elaborate example of conversion in "firewalls" mode. It illustrates the use of common arrays and of subprograms called both inside and outside the kernel. The Fortran 77 input program is shown in Figure 2; it is also on line under **`examples/cmax/`**.

include.inc

```
      INTEGER N, SQRTN
      PARAMETER (N = 100, SQRTN = 10)
      REAL COEFFICIENTS(N)
      COMMON /SHMOMMON/ COEFFICIENTS
```

main.f

```
      PROGRAM MAIN
      INCLUDE 'include.inc'
      REAL SNOW(N)

      CALL INPUT(SNOW)
      CALL MUNCH(SNOW)
      CALL OUTPUT(SNOW)
      END
```

input.f

```
      SUBROUTINE INPUT(A)
      INCLUDE 'include.inc'
      REAL A(SQRTN,SQRTN)

      DO J = 1,SQRTN
         DO I = 1,SQRTN
            A(I,J) = MOD(11 * I, 7) + MOD(15 * J, 9)
         END DO
      END DO
      DO I = 1,N
         COEFFICIENTS(I) = MOD(33 * I, 6)
      END DO
      CALL CRUNCH(A)
      END
```

program continues

Figure 2. An input program for conversion in "firewalls" mode.

Copyright © 1994 Thinking Machines Corporation

continued:

output.f

```
      SUBROUTINE OUTPUT(A)
      INCLUDE 'include.inc'
      REAL A(N)

      PRINT 10, A
 10   FORMAT(8F9.2)
      END
```

munch.f

```
      SUBROUTINE MUNCH(A)
      INCLUDE 'include.inc'
      REAL A(N)

      CALL GRIND(A)
      CALL CRUNCH(A)
      END
```

grind.f

```
      SUBROUTINE GRIND(A)
      INCLUDE 'include.inc'
      REAL A(N)

      DO I = 1,N
         IF (A(I) .GT. 0.0) THEN
            A(I) = COEFFICIENTS(I) * SQRT(A(I))
         ELSE
            A(I) = -1.0
         END IF
      END DO
      END
```

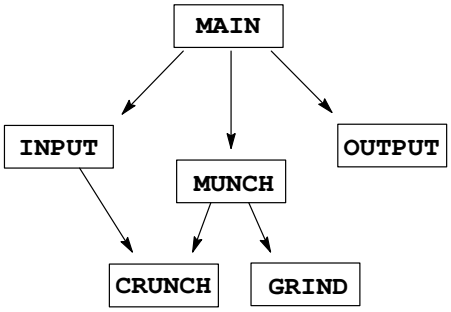
crunch.f

```
      SUBROUTINE CRUNCH(A)
      INCLUDE 'include.inc'
      REAL A(N)

      DO I = 1,N
         IF (A(I) .EQ. 0.0) A(I) = -1.0
      END DO
      END
```

Figure 2. Input program for conversion in "firewalls" mode *(cont'd)*.

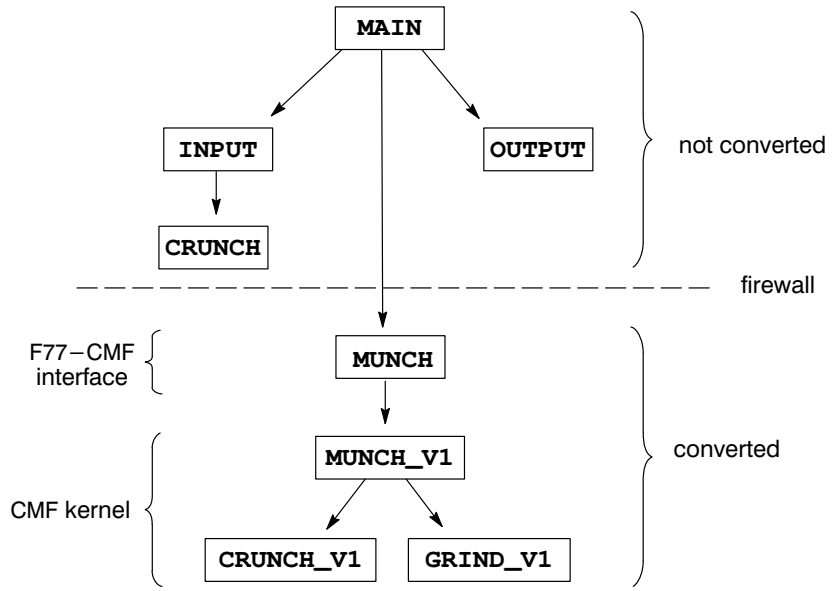
Copyright © 1994 Thinking Machines Corporation

The call tree for the program shown in Figure 2 is:



The computationally intensive kernel of the program is **MUNCH** and its callees **CRUNCH** and **GRIND**. This makes **MUNCH** a good choice as the interface routine. Note that **CRUNCH** is called both from **MUNCH** and from **INPUT**, which is outside the kernel.

After invoking CMAX with **-InterfaceRoutines=MUNCH**, we have a new call structure:



CRUNCH now appears twice but in different variants: in unconverted Fortran 77 form outside the kernel, as well as in CM Fortran form, **CRUNCH_V1**, within the kernel.

Copyright © 1994 Thinking Machines Corporation

MUNCH has been converted to two routines; the new routine **MUNCH** takes front-end arrays, creates CM arrays and transfers the data to them, calls the vectorized **MUNCH_V1**, and then copies the CM array data back to front-end arrays where necessary. If the original code in **munch.f** is:

```
      SUBROUTINE MUNCH(A)
      INCLUDE 'include.inc'
      REAL A(N)

      CALL GRIND(A)
      CALL CRUNCH(A)
      END
```

the converted output in **munch.fcm** is an interface routine **MUNCH** and a converted **MUNCH_V1**:

```
      SUBROUTINE MUNCH(A)
      INTEGER N           ! include file expanded
      PARAMETER (N=100)
      REAL A(N)
CMF$  LAYOUT A(:SERIAL)
      REAL A0(N)
CMF$  LAYOUT A0(:NEWS)
      INTEGER LL
      REAL COEFFICIENTS(N)
CMF$  LAYOUT COEFFICIENTS(:SERIAL)
      COMMON /SHMOMMON/COEFFICIENTS
      REAL COEFFICIENTS0(N)
CMF$  LAYOUT COEFFICIENTS0(:NEWS)
      COMMON /CM_SHMOMMON/COEFFICIENTS0 ! CM common block

      CALL CMF_FE_ARRAY_TO_CM(A0,A)
      CALL CMF_FE_ARRAY_TO_CM(COEFFICIENTS0,COEFFICIENTS)

      CALL MUNCH_V1(A0)

      CALL CMF_FE_ARRAY_FROM_CM(A,A0)
      END
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

```
      SUBROUTINE MUNCH_V1(A)
      INTEGER N, SQRTN
      PARAMETER (N = 100, SQRTN = 10)
      REAL COEFFICIENTS(N)
      COMMON /SHMOMMON/COEFFICIENTS0
      REAL A(N)
CMF$  LAYOUT A(:NEWS)
CMF$  LAYOUT COEFFICIENTS(:NEWS)
      REAL COEFFICIENTS0(N)
CMF$  LAYOUT COEFFICIENTS0(:SERIAL)
      COMMON /CM_SHMOMMON/COEFFICIENTS   ! CM common block

      CALL GRIND_V1(A)    ! Body of original MUNCH
      CALL CRUNCH_V1(A)
      END
```

The converted code for **MUNCH** contains declarations and statements to arrange for all the data needed in the parallel code below it (in **CRUNCH** and **GRIND**) to be moved from front-end arrays to CM arrays. The argument array **A** is given a corresponding CM array, **A0**. The front-end common array **COEFFICIENTS**, from the include file, is likewise given a corresponding CM common array, **COEFFICIENTS0**. Two calls to **CMF_FE_ARRAY_TO_CM** before the call to **MUNCH_V1** transfer the data to the CM arrays. Since the kernel modifies **A0** but not **COEFFICIENTS0**, only **A0** is transferred back with a call to **CMF_FE_ARRAY_FROM_CM**.

The converted subroutine **MUNCH_V1** declares its dummy argument and the common block as CM data, since they are used in array operations in its callees. Notice that **MUNCH_V1** uses the original common array name **COEFFICIENTS** to refer to the data in the CM common block **CM_SHMOMMON**. This trick reduces the amount of kernel code that needs to be changed.

Subroutine **INPUT**, which is outside the kernel, also takes array **A** as an argument, but notice what it does with it:

```
      SUBROUTINE INPUT(A)      ! A is 1D in caller
      INCLUDE 'include.inc'
      REAL A(SQRTN,SQRTN)       ! A is 2D in INPUT
      ...
      CALL CRUNCH(A)
      END
```


Copyright © 1994 Thinking Machines Corporation

The subroutine **INPUT** violates CM array argument-passing constraints by redeclaring its 1D actual argument as a 2D dummy argument. Since this subroutine does not reside in the kernel **MUNCH**, however, the data need not be constrained to a front-end home inside the firewall. The data in array **A** is transferred to CM array **A0** for use inside the kernel **MUNCH,** and the results later transferred back to **A**. Subroutine **INPUT** also passes array **A** on to **CRUNCH**, the unconverted version of **CRUNCH** that accepts a front-end array argument.

The code in **MUNCH_V1**, **CRUNCH_V1**, and **GRIND_V1** that operates on **A0** is fully vectorized, unconstrained by **A**'s front-end home in **INPUT**.

3.3 Using make to Convert and Compile

This section contains a sample Makefile for converting and compiling the program in Figure 2 with a firewall around the kernel **MUNCH**. The Makefile is in Figure 3 and also on line in **examples/cmax**.

One minor problem in CM Fortran 2.1 and earlier versions is that the main program unit of any program that contains CM Fortran code must be compiled with **cmf**. You cannot, therefore, compile the sample program with its original main routine in a **.f** file. You can rewrite the original **MAIN** as a subroutine and provide a dummy **.fcm** main program that calls it (as shown in Figure 4). You may also need a dummy **.f** main program that calls the subroutine **MAIN**, to be used when building a serial version of the program.

The Makefile uses the option setting **-OutputExtension=.cm.fcm** so that both **crunch.o**, created from the Fortran 77 version of **CRUNCH**, and **crunch.cm.o**, created from the converted **CRUNCH_V1**, can be linked into the executable. To avoid link-time errors, you must rename the output files for any routines that will continue to exist in unconverted **.f** files. If a program does not contain subroutines that are called both inside and outside the kernel, the change in output extension is not needed. It is wise precaution, however, to always change the output extension when converting in "firewalls" mode.

Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

```
.SUFFIXES: .f .o .cm.fcm .cm.o
# CMF definitions/defaults. Customize CMFFLAGS and LIBS.
CMF      =  cmf
OPT      =
DEBUG    =
CMFFLAGS =  -cm5 -vu $(OPT) $(DEBUG)
LIBS     =

# CMAX definitions and defaults. Customize as desired.
CMAX        =  cmax
CMAXFLAGS   =  -verbose=4 -list

# The package and the files it contains:
TARGET      =  myprog
PACKAGE     =  myprog
TRANSLATION =  myprog-cmaxed
ALLSRCS  =  main.f input.f output.f munch.f crunch.f grind.f
F77SRCS     =  main.f input.f output.f crunch.f
INTERFACE   =  munch
KERNEL      =  munch crunch grind
KERNELSRCS  =  $(KERNEL:%=%.f)
CMFSRCS  =  dummy-cmf-main.cm.fcm $(KERNELSRCS:%.f=%.cm.fcm)
OBJS     =  $(F77SRCS:%.f=%.o) $(CMFSRCS:%.cm.fcm=%.cm.o)

# The TRANSLATION (a dummy file) is used to keep track of
# how up-to-date the .fcm files are with respect to the .f
# files. The TARGET depends on both TRANSLATION and OBJS:

$(TARGET):  $(TRANSLATION) $(OBJS)
        $(CMF) $(CMFFLAGS) $(OBJS) $(LIBS) -o $(TARGET)

# If any .f file is changed, CMAX will be run and .fcm files
# recompiled if needed.

$(TRANSLATION) : $(ALLSRCS)
        $(CMAX) $(PACKAGE) -Add=$(ALLSRCS) -T \
            -Interface=$(INTERFACE) \
            $(CMAXFLAGS) -OutputExtension=.cm.fcm
        touch $@

# This rule tells how to compile the .fcm with cmf:
%.o: %.cm.fcm
        $(CMF) $(CMFFLAGS) -c $<
clean:
        rm -f $(TARGET) $(TRANSLATION) *.o *.lis *.ttab
        $(CMAX) $(PACKAGE) -remove
```

Figure 3. Sample `make` file for "firewalls" conversion and compilation.

Copyright © 1994 Thinking Machines Corporation

Rewrite `MAIN` as a subroutine:

`main.f`

```
      SUBROUTINE MAIN()
      INCLUDE 'include.inc'
      REAL SNOW(N)

      CALL INPUT(SNOW)
      CALL MUNCH(SNOW)
      CALL OUTPUT(SNOW)
      END
```

Have `make` choose one of these main programs:

`dummy-cmf-main.cm.fcm`

```
      PROGRAM DUMMY_CMF_MAIN
      CALL MAIN()
      END
```

`dummy-f77-main.f`

```
      PROGRAM DUMMY_F77_MAIN
      CALL MAIN()
      END
```

Figure 4. Multiple main routines for Makefile.

Copyright © 1994 Thinking Machines Corporation

4 Using the CMAX Library with CM Fortran 2.2 and Later

CM Fortran 2.2 has been enhanced to permit more convenient linking of libraries. If you use the dynamic allocation routines in the CMAX library, or if your converted program calls the utility procedure **CMF_SEND_***combiner*, you need to link with a platform-specific **CMAX** library.

Beginning with V2.2, the **cmf** compiler accepts **-lcmax** to indicate any of the CM-specific CMAX libraries. The compiler driver finds the library that is appropriate for the compilation's target platform, such as **libcmax_cm5_vu.a** or **libcmax_cm5_cmsim.a**. For example:

```
% cmf -cm5 -vu myprogram.fcm -lcmax
```

When linking Fortran 77 code that uses CMAX library features, you need to specify the serial version of the library on the **f77** command line. Its default location is **/usr/lib/libcmax.a**; check with your system administrator if this file has been moved.

Copyright © 1994 Thinking Machines Corporation