

Fortran 90 at NERSC:  
A Tutorial

Richard A. Gerber  
National Energy Research Scientific Computing Center  
ragerbernersc.gov

December 3, 1997

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Why F90? . . . . .	2
1.2	Cray's f90 compiler . . . . .	2
<b>2</b>	<b>Setting up your environment</b>	<b>2</b>
2.1	module command . . . . .	2
2.2	Available modules . . . . .	2
2.3	List loaded modules . . . . .	3
2.4	Load modules . . . . .	3
2.5	Show module effects . . . . .	4
<b>3</b>	<b>Compiling with f90</b>	<b>4</b>
3.1	Source file names . . . . .	4
3.2	Common command line options . . . . .	5
<b>4</b>	<b>Porting cf77 codes to f90</b>	<b>5</b>
4.1	Source code forms . . . . .	6
4.2	NAMELIST data file formats . . . . .	7
4.3	INTRINSIC functions . . . . .	8
4.4	Quick cf77 to f90 conversion . . . . .	10
<b>5</b>	<b>Special T3E considerations</b>	<b>11</b>
5.1	N\$PES . . . . .	11
5.2	MY_PE() . . . . .	11
5.3	DOUBLE PRECISION . . . . .	12
5.4	CRAFT . . . . .	12
5.5	Running programs . . . . .	12
5.6	Examples . . . . .	13
<b>6</b>	<b>Online Fortran 90 resources</b>	<b>14</b>
6.1	WWW pages . . . . .	14

## 1 Introduction

These documents describe the Fortran 90 programming environment on the Cray C90, J90 cluster and Cray T3E at NERSC. After reading these pages, you should be able to set up your UNICOS environment and compile and execute Fortran 90 code. This is not intended to be an introduction to the Fortran 90 language itself. General Fortran 90 links are included in the General Resources section.

### 1.1 Why F90?

Fortran 90 is the only FORTRAN compiler supported by Cray on any of the NERSC machines. In addition to being a richer and more powerful language than Fortran 77, Fortran 90 is compatible with the old F77 standards. Cray's f90 will compile almost all legal Fortran 77 code, and supports many new features.

### 1.2 Cray's f90 compiler

Cray's f90 compiler replaces the entire Cray cf77 compiling environment. There are no optimizing source code preprocessors; the compiler does all the optimization. When you invoke f90, you call the compiler and not a front end. Options to this command go straight to the compiler so there is no need to preface options with the `-wf` flag as you did with cf77.

## 2 Setting up your environment

The UNICOS `module` command configures your programming environment, ensuring that you are using the current version of the f90 compiler and support libraries. Invoke `module` before beginning your code development, compiling and testing.

### 2.1 module command

The `module` command is invoked at the UNICOS command prompt. The command `module load cf90` sets up the Fortran 90 compiling environment. Other modules can be used to define additional environments. For example the command `module load mpt` sets up the *Message Passing Toolkit* used by MPI and PVM message passing codes. Both of these packages *may* be loaded by default, but it is always a good idea to check. The online man page gives all the details; some common usages are described below.

### 2.2 Available modules

You can check to see which modules are available. For example, on the T3E (mcurie):

```
mcurie% module avail
```

```

----- /opt/modulefiles -----
CC                cf90                craytools          modules
CC.2.0.2.0        cf90.2.0.2.0          craytools.2.0.1.0 mpt
CC.2.0.2.2        cf90.2.0.2.1          craytools.2.0.1.1 mpt.1.1.0.0
PrgEnv            craylibs                craytools.2.0.2.0 mpt.1.1.0.0.4
cam               craylibs.2.0.1.0       cvt
cam.2.2.0.0.3     craylibs.2.0.2.0       cvt.3.1.0.0

----- /opt/modules/modules/modulefiles -----
CC                PrgEnv                craylibs.2.0.2.0   craytools.2.0.2.0
CC.2.0.2.2        craylibs                craytools           modules

```

## 2.3 List loaded modules

You can check to see which modules are currently loaded:

```
mcurie$ module list
```

```

Currently Loaded Modulefiles:
   1) craylibs      3) cf90             5) cam
   2) craytools    4) CC               6) PrgEnv
   7) mpt

```

## 2.4 Load modules

According to the output shown above, The f90 and mpt modules are loaded. If they were not, they could be loaded using

```
mcurie% module load cf90
mcurie% module load mpt
```

You can check the f90 compiler version with the command

```
mcurie% f90 -V
```

```
Cray CF90 Version 2.0.2.1 11/06/96 09:34:5
```

If, for some reason, you wanted to change to one of the other available versions, you would use

```
mcurie% module switch cf90 cf90.2.0.2.0
```

If you then check on the compiler version number, you will see that it has changed.

```
mcurie% f90 -V
```

```
Cray CF90 Version 2.0.2.0 11/06/96 09:35:57
```

This illustrates some of the power and convenience of the module command.

## 2.5 Show module effects

The effects of the module command can be seen using

```
mcurie$ module show cf90
```

```
-----  
/opt/modulefiles/cf90:  
  
Setenv CF90_T3E to /opt/ctl/cf90/cf90  
Prepend /opt/ctl/bin to PATH  
Prepend /opt/ctl/cf90/cf90/man to MANPATH  
Prepend /opt/ctl/cf90/cf90/nls/En/\%N.cat to NLSPATH  
Append /usr/man to MANPATH  
Prepend /opt/ctl/nls/En/\%N.cat to NLSPATH  
-----
```

## 3 Compiling with f90

The Cray f90 command compiles standard Fortran 90. A few of important conventions and common command line options are listed below. See the online man pages for further details.

### 3.1 Source file names

Fortran 90 source code file names **MUST** follow certain conventions or they **WILL NOT** compile correctly.

- filename.f : “Fixed Form” source code.
- filename.F : “Fixed Form” source code with gpp preprocessor.

The source code is formatted as Fortran 77 source code. For example, statements begin in column 7, characters in column 6 indicate a continuation line, and lines beginning with the letter C in column 1 are comments.

- filename.f90 : “Free Form” source code.
- filename.F90 : “Free Form” source code with gpp preprocessor.

The source code is formatted as standard Fortran 90 “Free Form” code. For example, statements may begin in any column, an ampersand (&) at the end of line indicates a continuation and ! indicates a comment.

### 3.2 Common command line options

Some of common options to the f90 compiler are given below. See the online man page for all the details.

- dp** Disables double precision arithmetic. Variables declared DOUBLE PRECISION, as well as double precision intrinsics, are converted to the default REAL type.
  - ep** Enables double precision arithmetic. Variables declared REAL, as well as real intrinsics, are converted to the DOUBLE PRECISION type. The Cray T3E does not support DOUBLE PRECISION.
  - f free** Force free source form.
  - f fixed** Force fixed source form.
  - I dirname** Search in directory *dirname* for include files.
  - l libname** Load library *libname*.
  - L libdir** Search directory *libdir* for libraries.
  - o execname** Give filename *execname* to binary executable.
  - c** Invokes only the compilation phase.
  - r listopt** Creates listing file. See online man pages for details.
  - G n** Debugging level, where *n* can be
    - 0** Full debugging; no optimization.
    - 1** Block by block debugging; retains some optimization.
  - R a** Run-time: compares number and type of arguments passed to a procedure with the number expected.
  - R b** Run-time: array bounds checking.
  - e v** Allocates variables to static storage.
  - O n** Optimization, where *n* can be
    - 0** No optimization.
    - 1** Conservative optimization: global scalar optimization.
    - 2** (T3E default) Moderate optimization: global scalar optimization, loop nest restructuring.
    - 3** Aggressive optimization: global scalar optimization, loop nest restructuring.
- One of many specific options** See online man pages.

## 4 Porting cf77 codes to f90

Existing cf77 codes should compile with little change under f90. Some of the common changes or modifications are described here.

## 4.1 Source code forms

Existing codes which use the old f77 “Fixed Form” source code format should compile under f90, as long as their filenames end with .f or .F.

An example of “Fixed Form” source code is

```
C
C File: nametest.f
C

      PROGRAM test_namelist

      IMPLICIT NONE

      REAL  a,b,c
      INTEGER ios

      NAMELIST / readme / a,b,c

      OPEN(UNIT=10,FILE='test_data')

      READ(UNIT=10,readme)
      CLOSE(UNIT=10)

      WRITE(UNIT=*, readme)

      STOP

      END
```

The same code in “Free Form” f90 might look like

```
!
! File: nametest.f90
!

PROGRAM test_namelist

  IMPLICIT NONE

  REAL :: a,b,c
  INTEGER :: ios

  NAMELIST / readme / a,b,c
```

```
OPEN(UNIT=10,FILE='test_data')

READ(UNIT=10,NML=readme,IOSTAT=ios)

IF(ios /= 0) THEN ! CHECK FOR ERROR READING FILE
  WRITE(UNIT=*,FMT='(A32)') 'Error reading nl file.'
  STOP
END IF

WRITE(UNIT=*, NML=readme)

STOP

END PROGRAM test_namelist
```

See the Resources section for pointers to online Fortran 90 resources.

## 4.2 NAMELIST data file formats

Some common NAMELIST input file formatting will not work with f90. Unlike with Fortran 77, the Fortran 90 standard specifies a format for NAMELIST input files. Some Cray cf77 extensions will not be accepted.

An example input file that worked under cf77 is

```
C
C Filename: test_data
C
C Here is my data. It's very important to understand
C what these numbers mean.!
C So it's good that I can put comments in like this.
C
C The &readme identifier below must
C not start in column 1
C
C I could have also used $readme if I had wanted to.
C
&readme
  a=1.0,
  b=2.0,
  c=3.0,
&end
```

The file above will not work under f90. The input needs to look very much like

```
&readme
    a=1.0,
    b=2.0,
    c=3.0
/
```

The important features are (1) the beginning `&`, (2) the ending `/`, (3) no comments allowed, and (4) no blank lines between the `&` and the `/`. The format shown is not required, for example everything could have been written on one line.

### 4.3 INTRINSIC functions

You are encouraged to use generic names for intrinsic functions under f90. For example `MIN(A,B)` works correctly whether `A` and `B` are `REAL` or `INTEGER`. The `MIN()` function returns a value that has same the type as its arguments.

Do not mix types of arguments. The following code worked under cf77, but **WILL NOT COMPILE** using f90.

```
C
C Filename: mintest.f
C
PROGRAM mintest

REAL*4 x,y

x = 2.0

y = MIN(x,1.0)

WRITE(6,*) y
STOP
END
```

In the above file, `x` is a `REAL*4`, but the compiler treats the constant `1.0` as the default `REAL`, i.e., `REAL*8`. The `MIN()` function expects the two arguments to be of the same type, so the compiler generates an error message and aborts compilation.

As an example, consider the following code that illustrates that the intrinsic function `SIN()` returns a different type, depending on its arguments.

```
PROGRAM sine_test
```

```
IMPLICIT NONE

REAL(KIND=4) :: a=1.1  !This is not very portable!
REAL(KIND=8) :: b=1.1  !But we're illustrating a point.
                    !KIND values are machine-dependent
                    !Here KIND=4 means 32-bit and
                    !    KIND=8 means 64-bit

PRINT *, 'The KIND of a is: ', KIND(a)
PRINT *, 'The KIND of b is: ', KIND(b)
PRINT *, 'The KIND of SIN(a) is: ', KIND(SIN(a))
PRINT *, 'The KIND of SIN(b) is: ', KIND(SIN(b))

STOP

END PROGRAM sine_test
```

The output from this program on the T3E is

```
The KIND of a is:  4
The KIND of b is:  8
The KIND of SIN(a) is:  4
The KIND of SIN(b) is:  8
```

## 4.4 Quick cf77 to f90 conversion

	<b>cf77</b>	<b>f90</b>
Compile and get executable.	cf77 -o a.out prog.f	f90 -o a.out prog.f
Only compile.	cf77 -c prog.f	f90 -c prog.f
Compile using static memory.	cf77 -c -a static prog.f	f90 -c -ev prog.f
Greatest level of autotasking	cf77 -Zp -c -prog.f	f90 -Otask3 -c prog.f
Greatest level of vectorization	cf77 -Zv -c prog.f	f90 -Otask0,scalar3,vector3 -c prog.f
Greatest level of optimization	cf77 -Zp -Zv -c -prog.f	f90 -Oscalar3,vector3,task3 -c prog.f
Compile using FPP (preprocessor)	cf77 -o a.out -Zu -Wl"libcon.a" prog.F  NOTE: -Zu is provided in CF77 to allow users to choose not to run FPP if they have already microtasked their codes.	f90 -o a.out -O3 -Wl"libcon.a" prog.F f90 -o a.out -O3 libcon.a prog.F  NOTE: There is no comparable F90 option to the -Zu option of CF77. NOTE: Not all CFPP\$ directive are not recognized by the F90 compiler. User should change CFPP\$ to CMIC\$.
Overindexing	-	f90 -c -Ooverindex prog.f.
Linking with library	cf77 -c prog.f segldr -o a.out \$IMSL	f90 -c prog.f segldr -o a.out \$IMSL  -or-  f90 -o a.out prog.f /usr/local/segdir/imsl  NOTE: You can do <i>echo \$IMSL</i> to find the full path to IMSL.

## 5 Special T3E considerations

The Cray T3E, [mcurie.nerisc.gov](http://mcurie.nerisc.gov), is a 512-node massively parallel processor (MPP). The machine is self-hosted; there is no front-end machine. When you log into the T3E you are logging into one of the Processing Elements (PE) or nodes. When you submit a multiprocessor job, each PE executes the same code.

Existing codes that use the MPI, PVM or SHMEM message passing libraries should compile with little change on the T3E. High Performance Fortran is available with PGHPF from the Portland Group. Since the T3E is a distributed memory machine, there is no possibility of autotasking.

There are special considerations needed for programming in f90 on the T3E. Some important ones follow.

### 5.1 N\$PES

You may wish to write your code so that it runs successfully on an arbitrary number of PEs without having to recompile it. A special runtime constant, `N$PES` is defined and can be used to determine the number of processors that are running the program.

For example an array can be allocated with the size based on the number of processors executing the program. Note that `N$PES` **CAN NOT** be used to define the size of an array in a `DIMENSION` statement.

```
PROGRAM npe

    IMPLICIT NONE
    INTEGER :: I
    REAL, DIMENSION(:), ALLOCATABLE :: A

    PRINT *, ' Number of processors is ', N$PES

    ALLOCATE(A(N$PES))

    DO I=1,N$PES
        A(I) = 1./FLOAT(I)
    END DO

    DEALLOCATE(A)

END PROGRAM npe
```

### 5.2 MY\_PE()

A special intrinsic function `MY_PE()` can be used to determine which PE is executing a certain portion of the code.

```

PROGRAM pe_number

    IMPLICIT NONE
    INTEGER :: ME
    INTEGER, INTRINSIC :: MY_PE

    ME = MY_PE()
    PRINT *, ' My processor number is ', ME

END PROGRAM pe_number

```

See the Resources section for pointers to online Fortran 90 resources.

### 5.3 DOUBLE PRECISION

Double precision arithmetic is deferred. If you attempt to declare variables as `DOUBLE PRECISION` you will get the message:

```

f90: WARNING SINE_TEST, File = sintest.f90, Line = 11, Column = 2
      DOUBLE PRECISION is not supported on this platform.
      REAL will be used.

```

### 5.4 CRAFT

Cray CRAFT directives are **NOT** supported in f90. They are, however, available for users of PGHPF.

### 5.5 Running programs

There are two ways to specify the number of PEs (`npes`) on which the program will run. You can choose `npes` at either compile time or at run time.

#### Selecting $N_{pes}$

To specify `npes` at compile time, use the `-X` option to `f90`:

```

mcurie% f90 -Xnpes -o filename.x filename.f90
mcurie% ./filename.x

```

where `npes` is the number of processor elements desired.

To specify `npes` at run time, use the `-Xm` compiler option to `f90` (or leave the `-X` option off since `-Xm` is the default). Then execute the program with the `mpprun` command:

```

mcurie% f90 -o filename.x filename.f90
mcurie% mpprun -nnpes filename.x

```

or

```
mcurie% f90 -Xm -o filename.x filename.f90
mcurie% mpprun -npes filename.x
```

## 5.6 Examples

Choosing npes at compile time:

```
mcurie% f90 -X 4 -o xnpe npe.f90
mcurie% ./xnpe
```

```
Number of processors is 4
Number of processors is 4
Number of processors is 4
Number of processors is 4
```

Since the code was compiled with the `-X 4` option, it can't be used with the `mpprun` command, as the following illustrates.

```
mcurie% mpprun -n 2 ./xnpe
```

```
mpprun: exec of './xnpe' failed:
Tried to set npes with
sitectl on a non-malleable a.out
```

Specify npes at run time:

```
mcurie% f90 -Xm -o xnpe npe.f90
mcurie% mpprun -n 8 ./xnpe
```

```
Number of processors is 8
Number of processors is 8
Number of processors is 8
Number of processors is 8
Number of processors is 8
Number of processors is 8
Number of processors is 8
Number of processors is 8
```

Since `npes` was not specified at compile time, the following example only runs on one processor, the one you're logged into interactively, which is shared with other users. This is probably not what you want.

```
mcurie% ./xnpe
```

```
Number of processors is 1
```

## **6 Online Fortran 90 resources**

### **6.1 WWW pages**

- CF90 Programming Environment
- Fortran 90 Essentials
- Fortran 90 Frequently Asked Questions
- Fortran 90: A Conversion Course for Fortran 77 Programmers
- Liverpool F90 courses
- Cray product info
- Fortran 90 Software Repository
- Fortran 90 for the Fortran 77 Programmer