Decoding and Converting Variant and Old Fortrans

A.k.a. Fortran Archaeology

Nick Maclaren

nmm1@cam.ac.uk

December 2006

Introduction

See "Introduction to Fortran Conversion"

This does NOT teach the new features
See "Introduction to Modern Fortran"
Even then, most details are only in books

This describes only what should be done
Starting from non–standard Fortran 90 code

What Have We Here?

Things that need changing, now
All are already hindering portability
Some will still work, sometimes . . .
Others are dead or almost totally dead

Here, I am heading backwards in time

Cray Pointers (1)

POINTER (<address>, <target>)

POINTER (LOCATION, ARRAY)
POINTER (LOCATION, ARRAY(0:M,0:N))

<target> is accessed like a normal array

<address> must be a scalar integer variable
Assigning to it causes ‘allocation’

Cray Pointers (2)

LOC(<data item>) gives address of any item
Sometimes CLOC, %LOC, SIZEOF and others

Almost always some allocation function
Quite often an interface to C malloc
Sometimes an ALLOC statement or similar

They are a nightmare for optimisation
http://portal.acm.org/)...
.../citation.cfm?id=140947.140948

Cray Pointers (3)

Innumerable, poorly documented variants

Can they point to CHARACTER data? How?
Derived types? Procedures? Arguments?
Can you perform address arithmetic?
What constraints on target's use?
What constraints on setting the address?
And more . . .

Cray Pointers (4)

- Convert to Fortran 90 strongly typed pointers

- Need to work out program's assumptions

Simple uses are easy to convert
Others can be harder, nasty or very nasty

Used to implement memory management
Even up to a compacting garbage collector

- Ask for advice if you have THAT problem

Other Current Extensions

Most extensions very rarely used or seen
Some existing relics are described later

Lots of non–standard library routines

- Chase up specification or ask for help

New Cray has BOOLEAN – don't ask
Actually a hack for system–dependent code

Late Fortran IV Relics

- Never part of any de jure standard

But Fortran IV was de facto standard
Each system had one or more variants of it

Will cover only most common issues

- Many have lasted well into modern era

Some are even still around, unfortunately

INTEGER*4, REAL*8 etc.

Length of datum – usually in bytes or words
Still accepted by many compilers
And still fairly common in some codes
Generally, INTEGER*4 = INTEGER
and REAL*8 = DOUBLE PRECISION

- Convert to KIND= parameterisation

NOT to INTEGER(4), REAL(KIND=8) etc.!

- Convert DFLOAT to DBLE

Function Definitions (1)

REAL FUNCTION FRED*8 (<args>)
CHARACTER FUNCTION JOE*22 (<args>)

Bizarre – I asked but never found out why!
Expected the following (but saw only once):

REAL*8 FUNCTION FRED (<args>)
CHARACTER*22 FUNCTION JOE (<args>)

The last line is valid Fortran 77 and beyond

Function Definitions (2)

No very nice way of cleaning this up, but isn't hard
The following is the 'best' (most modern) way

```
FUNCTION FRED (<args>)
REAL(KIND=KIND(0.0D0)) :: FRED

FUNCTION JOE (<args>)
CHARACTER(LEN=22) :: JOE
```

Function Definitions (3)

But all of the following are standard Fortran:

REAL(KIND=KIND(0.0D0)) FUNCTION FRED (<args>)
CHARACTER(LEN=22) FUNCTION JOE (<args>)

Or even (if you must):

CHARACTER*22 FUNCTION JOE (<args>)

Quadruple Precision

Usually REAL*16 and COMPLEX*32

Constants and data like 1.0Q0
Format elements like Q30.20
Functions like QSQRT, IQINT

- Convert to KIND= parameterisation

Change constants, functions in obvious way

DOUBLE COMPLEX etc. (1)

Nothing standardised, on political grounds

DOUBLE COMPLEX fairly widespread
Sometimes DOUBLE PRECISION COMPLEX
Much more often COMPLEX*16 etc.

Sometimes DOUBLE $\equiv$ DOUBLE PRECISION

- Convert to KIND= parameterisation

DOUBLE COMPLEX etc. (2)

- CMPLX(...) is a major Fortran ‘‘gotcha’’

It is NOT generic, but single precision

DCMPLX was D.P. version of CMPLX

- Replace by CMPLX(...,KIND=KIND(0.0D0))
- Or write your own – and declare it!

Also DIMAG, DREAL, CDSQRT, CDSIN etc.

Bit Mask Hacks

No bit mask operations in Fortran 66/77
Binary integers were not yet universal
Innumerable hacks to resolve this one

Some used INTEGER, some used LOGICAL
Sometimes <integer> .OR. <integer>
Sometimes integer arithmetic on LOGICAL

- Convert to INTEGER and modern functions
  IEOR etc. – see Fortran 90/95/2003

Error Handling / Debugging

‘D’ in column 1 meant only for debugging

- Replace by ’C...’, use Coco or remove

ERRSET, ERRTRA functions for error handling
Lots of other system–dependent hacks

- Generally, just disable them completely

Would be nice to convert to a modern use
But there isn’t one I can recommend

‘1’ As Last Dimension

A very heavily used trick/convention
Superseded by Fortran 77 – often still supported

```
SUBROUTINE FRED (A, N, M)
DIMENSION A(N,M,1)
```

Was often interpreted as equivalent to:

```
DIMENSION ARRAY(N,M,*)
```

- Just convert it to that!

I/O Extensions

Lots, for pre–Unix file system facilities
Keyed direct–access was fairly common

- Can often emulate facility, if necessary

Generally easy to recognise them
Working out their purpose is less easy
Ask for help or track down specification

You may see direct access with RECL=1
It usually meant some kind of byte stream mode
Fortran 2003 supports that properly

Other Features

Quite a few other dead/superseded features
Almost all pretty obvious in purpose
Portable ones mappable to Fortran 90
System–specific ones may be emulatable

Innumerable extra library routines, of course

- Seek advice if you can't decode them

Don't wait too long – or I may retire!

VAX/DEC Fortran

Fortran 77 extended almost beyond belief
Many features adopted in Fortran 90

Will describe most common extensions only
A few are shared by other extended Fortrans
Some were described as Fortran IV relics
Complete list would be a course in itself

Next 7 foils are all on this!

Tab Format Source

<label> <tab> <statement>
Allowed statements beyond column 72
[for fixed format source, too]
Subset of free format, so no problem
Except for continuation lines

<tab> <digit> <statement>

<tab>PRINT *,'This is a line
<tab>1that is continued'

Easy Changes

PARAMETER A=1 $\Rightarrow$ just add brackets
'123'O $\Rightarrow$ O'123', '1af'Z $\Rightarrow$ Z'1af'
.XOR. $\Rightarrow$ .NEQV.
Remove AUTOMATIC, STATIC $\Rightarrow$ SAVE
TYPE $\Rightarrow$ PRINT, ACCEPT $\Rightarrow$ READ

%VAL, %REF, %DESCR – remove and hope!
REWRITE is BACKSPACE+WRITE – clean up
SIND, TAND etc. take arguments in degrees

Recursion

Not often used, and not in VAX Fortran
May be indicated by use of AUTOMATIC

Declare all procedures in loop RECURSIVE
Check uses of SAVE and library calls

ENCODE and DECODE

Usually just an internal WRITE and READ

- Use an internal file instead

Also in some other Fortrans (e.g. CDC)
Not always with same specification
No, I can't remember the details!

STRUCTURE, UNION etc.

RECORD declares a STRUCTURE variable

- Convert them to a simple derived type

Convert a MAP to a simple derived type
UNION also to a derived type, but:
entries are pointers or allocatable

This will work only for clean uses!
UNION preserves non–overlaid data

Format Editing

‘$’ descriptor and ‘$’ or CHAR(0) control char.
Used for prompting – try nonadvancing I/O

Forms like ‘I’, ‘F.3’ use a default width

‘A’ can be used with any data type
Sometimes used for ‘unformatted’ I/O

And more . . .

Other Extensions

Can often be converted fairly easily
Nightmare if extended function is critical

FIND, DELETE, UNLOCK I/O statements
A zillion OPEN etc. options – see above

BYTE declaration means raw data

And there are others . . .

Old Cray Extensions

Old Cray (following CDC!) had quite a lot
Haven't been able to find a manual for details
Ask for help if you suspect Old Cray code
See later under CDC for possible issues

SHMEM facilities also started with Old Cray
Half–a–dozen very poorly–documented forms

Most are just one–sided message passing
Can usually replace by MPI – not always

Fortran 66/77 Relics

And some Fortran 66 era Fortran IV ones

These were dropped in Fortran 95/2003
Decremented/obsolescent in Fortran 90
You may see some of these
Many compilers have options to allow them

Most can be covered fairly briefly
Sane code is trivial to modernise

Automatic SAVE

Many compilers applied SAVE by default
And a great many programs assumed it!

Can declare all local variables SAVE
That is horribly unclean and inefficient

- Best solution is to clean up such code

Typically need to save only a few variables

DEFINE FILE

- Replace by OPEN (ACCESS='DIRECT')

DEFINE FILE unit (count, size, U, <var>)
System–dependent if size in words or bytes!
'U' for unformatted – very rarely 'F' or other
Associated variable <var> set to location

- READ(1'K) $\Rightarrow$ READ(1,REC=K) etc.

PAUSE

PAUSE <number or string>

Waited for the operator (!) to respond
Has been a simple diagnostic for decades
May still work in some compilers!

- Replace by PRINT or WRITE (*,*)

Assigned GOTO

ASSIGN <label> TO <integer variable>
GOTO <integer variable> [(<label>, ...)]
Allows statement label variables
Used for a variety of dirty tricks

- Restructure any such code completely

Very rarely could jump to calling routine
Probably only in some Fortran II compilers
Ask for help if you ever hit that one!

Assigned FORMAT Labels

ASSIGN <label> TO <integer variable>
READ (<integer variable>, ...)
WRITE (<integer variable>, ...)

Much cleaner but more rarely used

- Replace by format in CHARACTER variable

Same variable could be value, label or both
Sure sign of utterly insane programmer

Branching onto ENDIF

```
IF (...) GOTO 123
IF (...) THEN
...
123 ENDIF
```

- Just replace the ENDIF by:

```
ENDIF
123 CONTINUE
```

Floating-Point DO Parameters

```
DO 10 N = 0.1, 1.0, 0.1
10 CONTINUE
```

Er, rounding error, anyone?

- Replace the controls by integers

And then convert to the value you want

H Edit Descriptor

FORMAT (15HKilroy was here)

Comprehensible only to Fortran 66 people
‘15’ is the number of chars after the ‘H’

- Just convert to:

FORMAT (’Kilroy was here’)

Genuinely Old Codes

Fortran 66, II and their variants
Some were still being run up to c. 1990
With all compiler ‘compatibility’ options on

- Consider whether they are worth fixing

May be better to rewrite from design
Or even redesign from scratch

Extended Fortran 66 Languages

Far too many to describe or remember
Often introduced ‘structured’ coding
Blocks, while loops, derived types etc.
Often preprocessors written in Fortran

Most died when Fortran 77 arrived
Even when not superseded until Fortran 90
Ask for help if you come across one

The main exception is ratfor

Ratfor/Ratfor77

Attempt to make Fortran look like C
Religious exercise by AT&T camp
Made headway among ex computer scientists

It made some good points about Fortran
And it did add some good features
But put people off by its fanaticism

Ratfor/Ratfor77 $\Rightarrow$ Fortran 90 converter

- Search for ratfor90 – it is a Perl program

Example Ratfor Syntax

if (...) <statement> else <statement>
while (...) <statement>
repeat <statement> until (...)
for (. . . ; . . . ; . . .) <statement>
switch (<integer expression>) {
    case <value>[,<value>]: <statement>
    default: <statement>
}
define <name> <expression>
8%77, 16%2ff are octal and hexadecimal

Dirty Tricks

Several widespread dirty tricks
Some compilers/libraries were severely hacked

- Very strange code may indicate these

- Some distinguished zero from blank fields

Rarely, missing values or uninitialised
Often used the sign of zero to test them

Blank common sometimes could be expanded

- Often used as workspace for that reason

CDC Fortran

Quite a few extensions – some got into Fortran 77
The VAX Fortran of its day, though not as bad

Names could be 7 characters long on CDC 7600!

Extra Hollerith – e.g. 15Lleft–justified, 5Rright

Lots of abbreviations for LOGICAL – e.g. .T., .A.
Could mix numeric and logical almost *a*d lib!
used for masking, in a very C–like fashion

Dead Fortran 66 Features

They were dropped in Fortran 77
You may occasionally see a few of them
Some compilers still support one or two

Most indicate truly revolting code
Generally best rewritten from scratch
Using Fortran 66 compiler options is BAD sign

- Best to seek advice if you hit these

Array Dimensions

```
DOUBLE PRECISION X(10,20), A
A = X(15,5)
```

Legal in Fortran 66, but not afterwards

- Still assumed in some FFT codes!

```
DOUBLE PRECISION X(10,20), Y(200)
EQUIVALENCE (X(1),Y(1))
```

But avoid EQUIVALENCE, anyway!

Character Data Problems

No character data type at all in Fortran 66
Hollerith constants (next slide) accepted

- Numeric types used to hold characters

Techniques too complex to describe here
Overflow, normalisation, comparisons . . .

- Ask for help if you hit such code

Hollerith Constants etc.

As in FORMAT: nHstring – e.g. 4Hyuck
Allowed in arguments and DATA statements
Some compilers allowed them in assignments

Allowed to read INTO strings in formats
Formats could be arrays of any type

Rewrite any such code using CHARACTER

Extended Ranges

Could branch out of innermost DO loop
And then (legally) branch back
IF no active loop parameters updated
and some other restrictions obeyed!

Restructure any such code from scratch
Using it was always bad practice
Often done to save cost of function call

One-Trip DO Loops

```
K = 9
DO 100 J = 10, K
100 CONTINUE
```

Went through once with value 10

Rewrite any such code from scratch
Assuming it was bad practice even in 1970!
Many compilers still have an option

Other Fortran 66 Features

Too many, obscure and horrible to list
Mostly errors not explicitly forbidden
I have very rarely seen any of these
Except in programs that needed rewriting

- And a lot fewer intrinsic functions

E.g. LEN(x) was an external function call

Fix the very obvious oversights/changes
Rewrite unspeakable code completely

Fortran II Features

You probably will never see these
But some lingered for many decades
I used a compiler with two in 1998!

I will just mention them briefly

Fortran II Long-lived Relics

PUNCH statement
Like PRINT, but secondary unit
Convert to WRITE statement

ABSF/XINTF/MAX0F/FLOATF/etc. functions
Convert to modern generic names

CALL SLITE / SLITET
Later sense light emulation functions
They just set and test a few global bits

Real Fortran II Statements

READ INPUT TAPE
WRITE OUTPUT TAPE
READ TAPE (and WRITE TAPE)
READ DRUM (and WRITE DRUM)
IF (SENSE LIGHT) ...
IF (SENSE SWITCH) ...
IF DIVIDE CHECK ...
IF [condition] OVERFLOW ...

Give the program a decent burial
It hasn't been used in over 35 years . . .

Egtran (Fortran II on KDF9)

Recursion, asynchronous I/O etc.
25 years ahead of Fortran 90!

There were other extended Fortran IIs
E.g. use of assigned GOTO mentioned above

Some ancient codes look strangely modern
Please do show me any interesting ones