Fortran90/5:

A Bridge Between the Past and the Future

Matthew Segall
University of Cambridge

Overview

- Why Fortran?
- Fortran90/5 is a modern language!
- ‘Nice’ features of Fortran90/5
- Optimisation of Fortran
- Potential pitfalls with Fortran90/5
- Conclusion

Science or Theology?

If I were to build a bridge, I would seriously consider what material to build it out of. Also, the design of bridge would be heavily influenced by the choice of material and vice versa.

… To choose a language for some software, you need knowledge of several languages, and to design a piece of software successfully, you need a fairly detailed knowledge of the chosen language – even if you never personally write a single line of that software

- Stroustrup

Fortran History

- FORTRAN, for FORmula TRANslation was developed by IBM in 1954 as an alternative to assembler
- Major versions: FORTRAN IV (1961), FORTRAN66, …77,…90, …95 and now …2003
- Designed for numerical work. More flexible now than ever before, but still not a 'general purpose' language

What is the Right Tool?

Task	Language
Manipulating complex data	C++?
Platform independent GUI for client-server platform	Java?
Post-processing of textual output	Perl/Python?
High performance numerical simulation	???

Not C(++)???

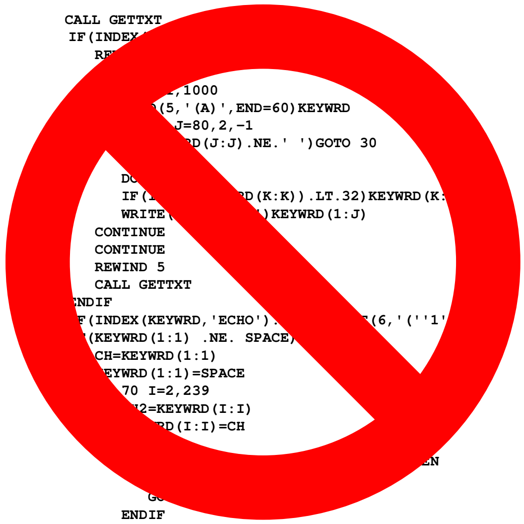
- C was designed as a general purpose language with "economy of expression" and "absence of restriction"

  - K&R preface

- "Neither C nor C++ were designed primarily with numerical computation in mind"

  - Stroustrup, *C++ Programming Language,* 3rd Edition

Fortran77



```
      CALL GETTXT
       IF(INDEX
          RE
              ,1000
             (5,'(A)',END=60)KEYWRD
              J=80,2,-1
             D(J:J).NE.' ')GOTO 30

             DO
             IF(I         D(K:K)).LT.32)KEYWRD(K:
             WRITE(          )KEYWRD(1:J)
          CONTINUE
          CONTINUE
          REWIND 5
          CALL GETTXT
       NDIF
       F(INDEX(KEYWRD,'ECHO').          (6,'(''1'
        (KEYWRD(1:1) .NE. SPACE)
          CH=KEYWRD(1:1)
          EYWRD(1:1)=SPACE
             70 I=2,239
              2=KEYWRD(I:I)
              D(I:I)=CH

                                                EN

               G
             ENDIF
 70       CONTINUE
```

Dr. Matthew D. Segall

University of Cambridge

Fortran90

```
if (keyword_present(ikey_comment)) then
    call io_freeform_string(keywords(ikey_comment)%key,comment)
end if

iprint = 1   ! verbosity control

if (keyword_present(ikey_iprint)) then
    call io_freeform_integer(keywords(ikey_iprint)%key,iprint)
    select case (iprint)
          case (:0)
             iprint=0
          case (3:)
             iprint=3
    end select
end if

! Over-determination cross-check

if (keyword_present(ikey_continuation) .and. keyword_present(ikey_reuse)) then
    mesg='duplication with keywords ' // trim(keywords(ikey_continuation)%key) &
          & // ' and' // trim(keywords(ikey_reuse)%key)
    call parameters_error_log(mesg)
end if
```

Dr. Matthew D. Segall

University of Cambridge

‘Modern’ Features

Feature	
Free format source code	✓
Dynamic memory allocation	✓
Derived types	✓
Subroutine/function/operator overloading	✓
Modularity (but not OO)	✓

Modules

```
module example_module

private ! Default is private

real, public :: externally_visible
integer, dimension(100), public :: extern_int_array

character(len=25) :: internal_use_only

public :: my_external_sub

contains

subroutine my_external_sub(arg1,arg2,...)
...
end subroutine

subroutine my_private_routine(arg1,arg3,...)
...
end subroutine

end module
```

Private variables and subroutines

Public variables. No more common blocks!

Public subroutines. ‘methods’

Memory Allocation

```
subroutine demonstrate_allocate(N,M)
integer :: N,M

real, dimension(:), allocatable :: vector
real, dimension(:,:), allocatable :: two_dimensional

allocate(vector(N))
allocate(two_dimensional(N,M))

...

if (.not.allocated(vector)) deallocate(vector)
deallocate(two_dimensional)
...
end subroutine demonstrate_allocate
```

Allocatable declaration.

Allocation of memory

Test if array has been allocated

Don't forget to deallocate memory when done!

Dr. Matthew D. Segall

University of Cambridge

Derived Types

```
type wavefunction

integer :: num_basis_functions
integer :: num_bands
integer :: num_kpts

complex, dimension(:,:,:), allocatable :: coefficients

end type wavefunction

program wavefunction_example

type(wavefunction) :: ground_state

ground_state%num_basis_functions = 100
ground_state%num_bands = 10
ground_state%num_kpts = 5

allocate(ground_state%coefficients(100,10,5))

call calculate_gs(ground_state)

...
end program wavefunction_example
```

Group associated variables.

Allocatable in F95. Only pointers in F90.

Pass types variables to simplify calls

Dr. Matthew D. Segall

University of Cambridge

Array Syntax

```
real, dimension(100,100) :: A
real, dimension(10,10) :: B
real, dimension(1000) :: x,y,z

A = 0.0d0

...

A(1:10,1:10) = B

...

z = x + y
```

Assign 0.0d0 to every element of A

Copy B into top corner of A

Vector addition

Interfaces (prototypes)

```
interface
   real function dot(A,B)
        real, dimension(:) :: A,B
   end function dot
end interface

real :: result
real, dimension(100) :: x,y
complex, dimension(100) :: s,t

result = dot(x,y)

result = dot(s,t)
```

Defines type and arguments of function

This call is fine.

This call will cause an error at compile time

Dr. Matthew D. Segall
University of Cambridge

Interfaces (overloading)

```
interface dot

   real function real_dot(A,B)
        real, dimension(:) :: A,B
   end function real_dot

   complex function cmplx_dot(A,B)
        complex, dimension(:) :: A,B
   end function cmplx_dot

end interface dot

real :: result
real, dimension(100) :: x,y
complex, dimension(100) :: s,t
complex :: cmplx_result

result = dot(x,y)

cmplx_result = dot(s,t)
```

Defines overloaded definitions of dot

This is now OK.

Implicit size arrays

```
Subroutine dimensions_example(A,B,C)

   real, dimension(*) :: A
   real, dimension(:) :: B
   real, dimension(:,:) :: C
   integer :: dim1,dim2

   dim1 = size(B)

   dim2 = size(A,2)

   do i=1,dim1
        do j=1,dim2
                C(I,j) = A(j)*B(i)
        end do
   end do

end subroutine dimensions_example
```

Old-fashioned implicit shape

Implicit size

Find dimensions

intent

```
subroutine example_intent(A,B,x,y)

   real, dimension(:), intent(in) :: A
   real, dimension(:), intent(out) :: B
   real, intent(out) :: x
   real, intent(inout) :: y

   B = A*y

   A = A*y

   B = A*x

   y = A(1)*y

end subroutine example_intent
```

Contents will not change in subroutine

Data on entry is not used

Used for both input and output

Pure Functions and forall (F95)

```
pure real function example_pure(x,y)

   return y*sqrt(x)

end function example_pure

program test

   real, dimension(50) :: A,B,C
   integer :: i
...
   forall (i=1:50)

       A(i)=example_pure(B(i),C(i))
   end forall
...
end test
```

A pure function can have no side effects

Hence iterations of this loop are completely independent

Intrinsics

Lots of useful intrinsics, e.g.

Function	Description
dot_product(A,B)	Returns dot product of vector arguments
matmul(A,B)	Returns matrix product of matrix arguments
transpose(A)	Returns transpose of matrix A
minval(A)	Returns minimum element of array A
maxloc(A)	Returns index of maximum element of array A

Putting it together

```
Module interval
   type interva
        real :
   end type int
   interface op
        module
   end interfac
contains
   function add
        type(i
        type(i

        add_in
        add_intervals%upper = A%upper + B%upper
   end function add_intervals
end module interval_arithmetic
```

```
program example
        use interval_arithmetic

        type(interval) :: int1, int2, int3

        int1%lower = 1.0
        int1%upper = 5.0

        int2%lower = 4.5
        int2%upper = 10.3


        int3 = int1 + int2

        ...

end program exmaple
```

Optimisation: Do/For

In C/C++:

```
for (i=0;i<N,i++) {
   ...
   /* Do some stuff */
   ...
}
```

'Stuff' can modify the value of i

In Fortran:

```
do i=1,N
   ...
   ! Do some stuff
   ...
end do
```

'Stuff' **cannot** modify the value of i

Opt.: Independence

In C:

```
void vadd(int N, double *a,
                 double *b, double *c)
{
   int i;
   for(i=0;i<N;i++) c[i]=a[i]+b[i];
}
```

What if this is called as:

```
double *x;
X[0]=1 ; x[1]=1;
vadd(N-2,x,x+1,x+2);
```

Opt.: Independence

In Fortran:

```
subroutine vadd(A,B,C)
   real, dimension(:) :: A,B,C
   integer :: i

   do i=1,size(A)

        C(i)=A(i)+B(i)
   end do

end subroutine vadd
```

In Fortran, arguments cannot be aliased

So, the iterations of this loop **must** be independent

Dr. Matthew D. Segall

University of Cambridge

Beware!

```
! Finding the trace of a sub-
  array
real, dimension(N,N) :: A
! Pass a sub-array of A of
  dimension m
tr = bad_trace(A(1:m,1:m),m)
!Pass the whole array with
  dimension of sub-array
tr = good_trace(A(1:m,1:m))
```

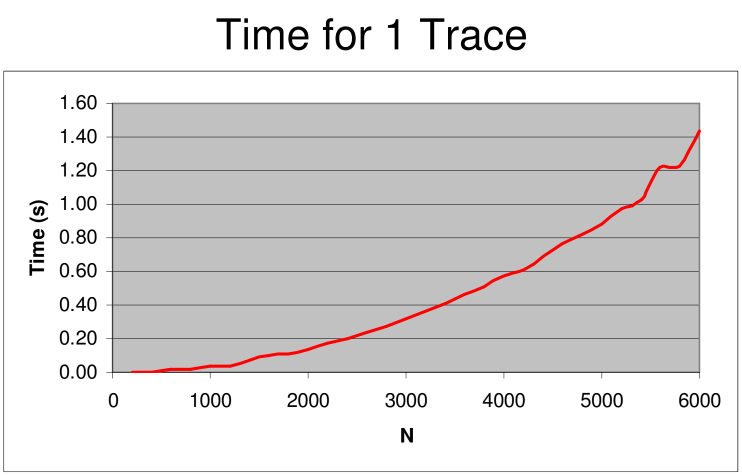
Dr. Matthew D. Segall

University of Cambridge

bad_trace

```
real function bad_trace(A,n)
integer :: n
real, dimension(n,n) :: A
integer :: i

bad_trace=0.0
do i=1,n
bad_trace=bad_trace+A(i,i)
end do

return
end function bad_trace
```

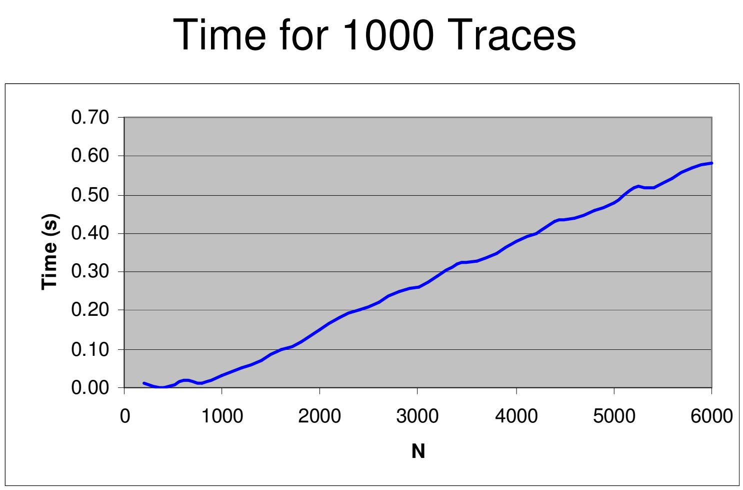

bad_trace times



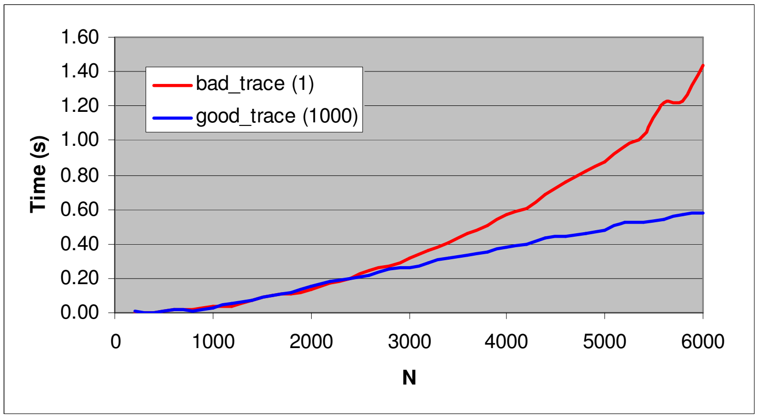
good_trace

```
real function good_trace(A)
real, dimension(:,:) :: A
integer :: i

bad_trace=0.0
do i=1,size(A,1)
bad_trace=bad_trace+A(i,i)
end do

return
end subroutine good_trace
```


good_trace times



Comparison



Dr. Matthew D. Segall

University of Cambridge

What's Optimal?

- Consider the following 5 fragments of code performing the the matrix multiplication **D** = **ABC**

Dr. Matthew D. Segall

University of Cambridge

Explicit loops

```
do i=1,N
   do j=1,N
        rtemp=0.0d0
        do k=1,N
                rtemp=rtemp + A(i,k)*B(k,j)
        end do
        temp(i,j)=rtemp
   end do
end do
do i=1,N
   do j=1,N
        rtemp=0.0d0
        do k=1,N
                rtemp=rtemp + temp(i,k)*C(k,j)
        end do
        D(i,j)=rtemp
   end do
end do
```


Explicit loops 2

```
do i=1,N
   do j=1,N
        temp(i,j)=0.0d0
        do k=1,N
                temp(i,j)=temp(i,j) + A(i,k)*B(k,j)
        end do
   end do
end do

do i=1,N
   do j=1,N
        D(I,j)=0.0d0
        do k=1,N
                D(i,j)=D(i,j) + temp(i,k)*C(k,j)
        end do
   end do
end do
```

Fortran Intrinsic

```
temp = matmul(A,B)

D=matmul(temp,C)
```

Dr. Matthew D. Segall

University of Cambridge

Fortran Intrinsic 2

```
D=matmul(matmul(A,B),C)
```

Dr. Matthew D. Segall

University of Cambridge

BLAS Library

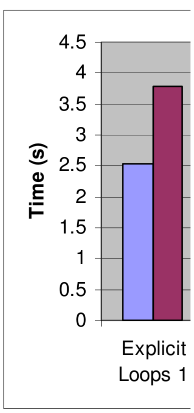
```
call dgemm('N','N',N,N,N,1.0d0,A,N,B,N,0.0d0,temp,N)

call dgemm('N','N',N,N,N,1.0d0,temp,N,C,N,0.0d0,D,N)
```

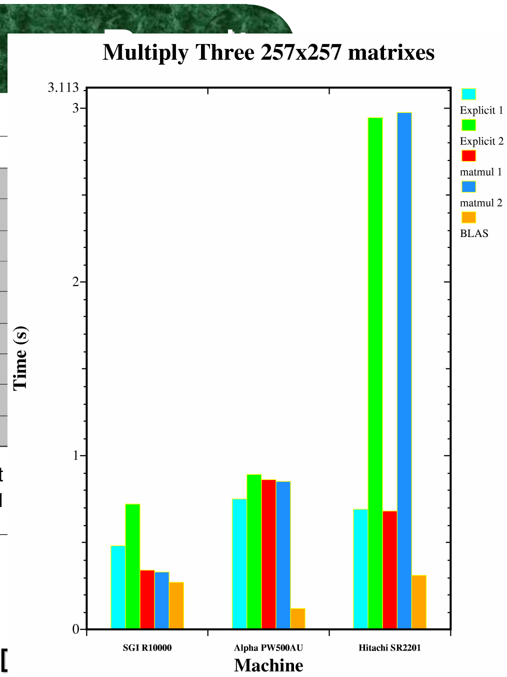
Dr. Matthew D. Segall

University of Cambridge

---

ension 257





Dr. Matthew D

of Cambridge

Conclusions

- Fortran is **not** a general-purpose language, but has been designed for numerical efficiency
- Fortran77 is long dead. Fortran90/95 is a modern language
- No matter what language
  - The 'fanciest' features are usually sub-optimal
  - You need to understand the specification to write optimal code
- **Use the right tool for the job... Whatever that may be!**