Introduction to Fortran Programming

Kanglin Chen

14. April 2009

What is Fortran?

What is Fortran?

A team lead by John Backus developed Fortran, FORmula TRANslation System, in 1954, one of the earliest high-level languages.

What is Fortran?

A team lead by John Backus developed Fortran, FORmula TRANslation System, in 1954, one of the earliest high-level languages.

Development:

- Fortran 66: The first standard for a programming language in 1966.
- Fortran 77: New standard in 1978.
- Fortran 90/95: Now.

Why is Fortran?

Why is Fortran?

functionality	F77	C	C++	F90
numerical robustness	2	4	3	1
data parallelism	3	3	3	1
object oriented programming	4	3	1	2

Why is Fortran?

functionality	F77	C	C++	F90
numerical robustness	2	4	3	1
data parallelism	3	3	3	1
object oriented programming	4	3	1	2

Fortran is a scientific program language widely used by scientists and engineers. Compared to C/C++ there's another advantage for scientic computing: the imaginary unit i is predefined.

Subprograms

function Example

```
real function maximum(iarray, nsize)

implicit none

integer, intent(in) :: nsize
real, dimension(nsize), intent(in) :: iarray
integer :: j

maximum = iarray(1)

do j = 2,nsize
if (max.LT.iarray(j)) maximum = iarray(j)
end do

end function maximum
```

Subprograms

subroutine Example

```
subroutine maxmin(iarray, nsize, max, min)

implicit none

integer, intent(in) :: nsize
real, dimension(nsize), intent(in) :: iarray
real, intent(out) :: max, min
integer :: j


max = iarray(1)
min = iarray(1)

do j = 2,nsize
if (max.LT.iarray(j)) max = iarray(j)
if (min.GT.iarray(j)) min = iarray(j)
end do

end subroutine maxmin
```

implicit none

Strong typing: all typed entities must have their types specified explicitly.

By default an entity in Fortran that has not been assigned a type is implicitly typed, e.g. entities that begin with i,j are of type integer $\longrightarrow$ dangerous source of errors.

The statement implicit none turns on strong typing and its use is strongly recommended.

Module & allocatable array

```
module constants

implicit none

real, parameter :: pi = 3.14159265358979324

real,allocatable,dimension (:) :: iarray

end module constants


program module_example

use constants

implicit none

integer :: r

r = 1

allocate(iarray(2))

iarray = (/2*pi*r, pi*r*r/)

end program module_example
```

Compiler

gfortran is the Fortran 95 compiler that is part of GCC (the GNU Compiler Collection). It has replaced the g77 compiler.

ifort is the Intel Fortran Compiler. It is up to now the most used fortran compiler in Windows and Linux. E.g. in Linux:

Compiler

gfortran is the Fortran 95 compiler that is part of GCC (the GNU Compiler Collection). It has replaced the g77 compiler.

ifort is the Intel Fortran Compiler. It is up to now the most used fortran compiler in Windows and Linux. E.g. in Linux:

- *ifort bsp.f90*

Compile the source code into an executable data *a.out*.

- *ifort -o bsp bsp.f90*

Compile the source code into an executable data *bsp*.

- *ifort -o bsp bsp1.f90 bsp2.f90*

Compile several source codes and link to an executable data *bsp*.

Makefile

For a project it is always convenient to use a makefile to add some compiler options and compile the project.

```
FC = ifort -integer-size 64 -real-size 64 -double-size 128 -axT -xT -vec-report0 -openmp


objects =   main.o area_normal.o dE2.o E_poyn.o orthonorm.o scalarprod_dist.o



sources=   main.f90 area_normal.f90 dE2.f90 E_poyn.f90 orthonorm.f90 scalarprod_dist.f90



streuprojekt :  $(objects)
$(FC) -o streuprojekt $(objects)

main.o :  $(sources)
$(FC) $(sources) -c

clean :
rm streuprojekt $(objects)
```

-o specifies the name for an output file

-c causes the compiler to compile to an object (.o) file

IMSL

The IMSL Fortran Numerical Library is a comprehensive library of mathematical and statistical algorithmus for fortran developers in one package. The IMSL is highly accurate and reliable. It contains mainly the following categories:

1. Math/Library
   Linear Systems, Interpolation and Approximation, Differential Equations, Optimization...
2. Special Functions
   Elementary Functions, Gamma Functions, Elliptic Integrals...
3. Stat/Library
   Regression, Correlation, Sampling...

IMSL is also available for Java, C/C++.

Using Mex-Files to call Fortran Programs from Matlab

You can call Fortran subroutines from the Matlab command line as if they were built-in functions. These programs are called binary Mex-Files.

Translate the subroutine into a fortran mex-file routine:

```
subroutine mexFunction(nlhs, plhs, nrhs, prhs)
      integer nlhs, nrhs
      mxpointer plhs(*), prhs(*)
```

Parameter	Description
prhs/plhs	An array of input/output arguments.
nrhs/nlhs	The size of prhs/plhs arrays.

```
subroutine mexFunction(nlhs, plhs, nrhs, prhs)
  use mexf90                    ! API function definitions

  implicit none

  integer, intent(in) :: nlhs, nrhs
  integer, intent(in), dimension(*) :: prhs
  integer, intent(out), dimension(*) :: plhs

  integer :: m,n
  integer, pointer :: A,B       ! These are the pointers to the matrix data
  double :: alpha

  ! Get data and size of the input matrix
  A => mxGetPr(prhs(1))
  m = mxGetM(prhs(1))
  n = mxGetN(prhs(1))

  ! Get scalar value
  alpha = mxGetScalar(prhs(2))

  ! Create output matrix
  plhs(1) = mxCreateDoubleMatrix(m,n,0)
  B => mxGetPr(plhs(1))

  ! Call subroutine for multiplication
  call scalarMult(A,B,alpha,m,n)

end subroutine mexFunction

subroutine scalarMult(A,B,alpha,m,n)
```

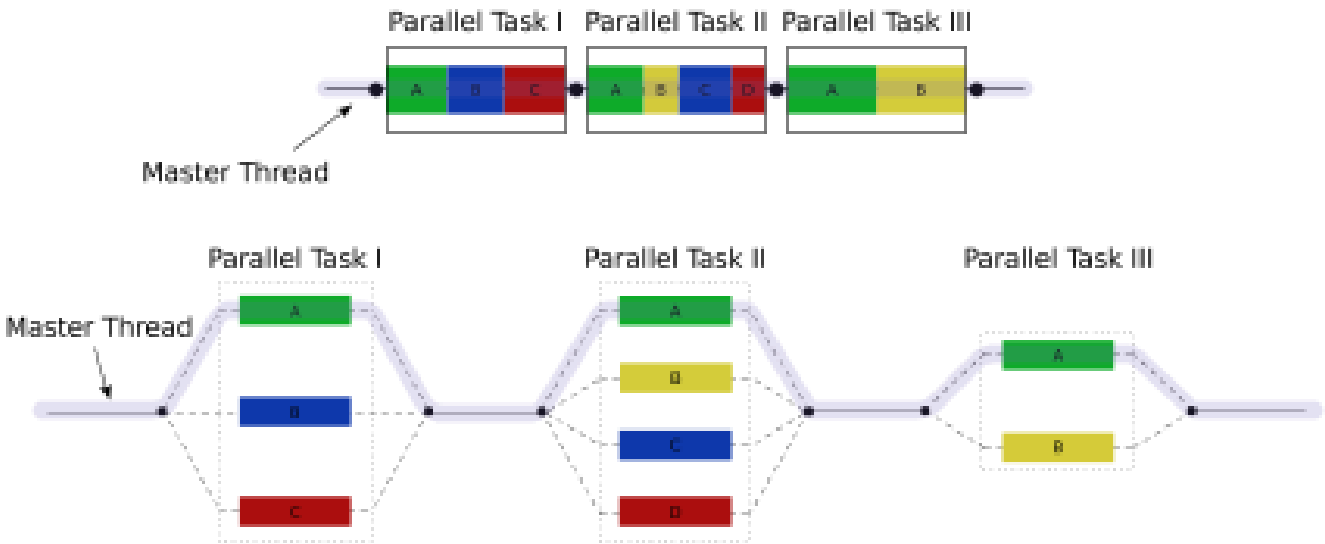
Parallelization with Openmp

The Openmp(Open Multi-Processing) is an application programming interface that supports multi-platform shared memory multiprocessing programming.

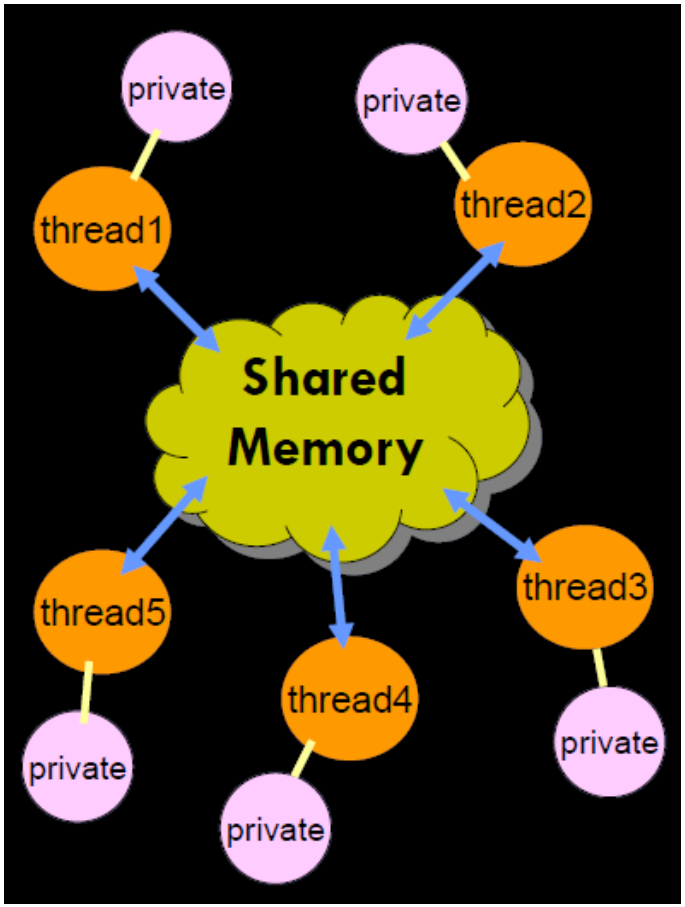
Parallelization with Openmp

The Openmp(Open Multi-Processing) is an application programming interface that supports multi-platform shared memory multiprocessing programming.

The Fork-Join-Prinzip:



Parallelization with Openmp



Parallelization with Openmp

```
program bsp
  use omp_lib

  implicit none

  integer :: thread_nr

  ! fork
  !$omp parallel num_threads(3)
    thread_nr = omp_get_thread_num()

    ! For master thread
    if( thread_nr == 0 ) then
      write( *, * ) 'Insgesamt gibt es ', omp_get_num_threads(), 'Thread(s)'
    end if

    ! Output for other threads
    write( *, * ) 'Thread ', thread_nr, 'ist aktiv'
  ! join
  !$omp end parallel

! Ausgabe:
!    Insgesamt gibt es            3 Thread(s)
!    Thread            0 ist aktiv
!    Thread            1 ist aktiv
!    Thread            2 ist aktiv
end program bsp
```

Parallelization with Openmp

```
program bsp
  use omp_lib

  implicit none

  integer :: i, tnr

  call omp_set_num_threads( 3 )

  !$omp parallel private( i )
    !$omp do
      do i = 1, 20
        tnr = omp_get_thread_num()  ! Aktuelle Threadnummer
        write( *, * ) 'Thread', tnr, ':',  i
      end do
    !$omp end do
  !$omp end parallel
end program bsp
```

Thank you for your attention!