CEE 618 Scientific Parallel Computing (Lecture 6): Object Oriented Programming (OOP) of FORTRAN 2003 and OpenMP vs. MPI

Albert S. Kim

Department of Civil and Environmental Engineering
University of Hawai'i at Manoa
2540 Dole Street, Holmes 383, Honolulu, Hawaii 96822

CEE 618 Scientific Parallel Computing (Lecture 6): Object Oriented Programming (OOP) of FORTRAN 2003 and OpenMP vs. MPI

Albert S. Kim

Department of Civil and Environmental Engineering
University of Hawai'i at Manoa
2540 Dole Street, Holmes 383, Honolulu, Hawaii 96822

ssh fractal - ssh c18.

CEE 618 Scientific Parallel Computing (Lecture 6): Object Oriented Programming (OOP) of FORTRAN 2003 and OpenMP vs. MPI

Albert S. Kim

Department of Civil and Environmental Engineering
University of Hawai'i at Manoa
2540 Dole Street, Holmes 383, Honolulu, Hawaii 96822

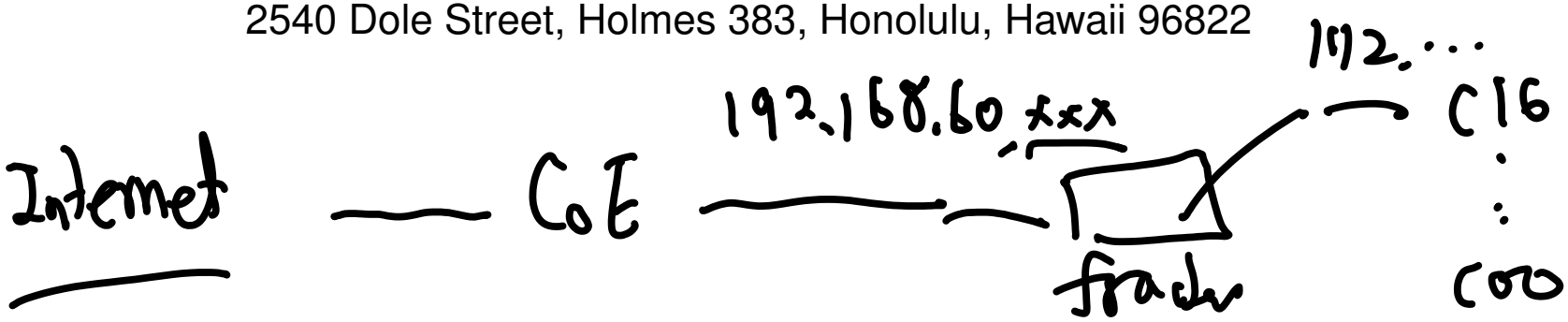


Table of Contents

Outline

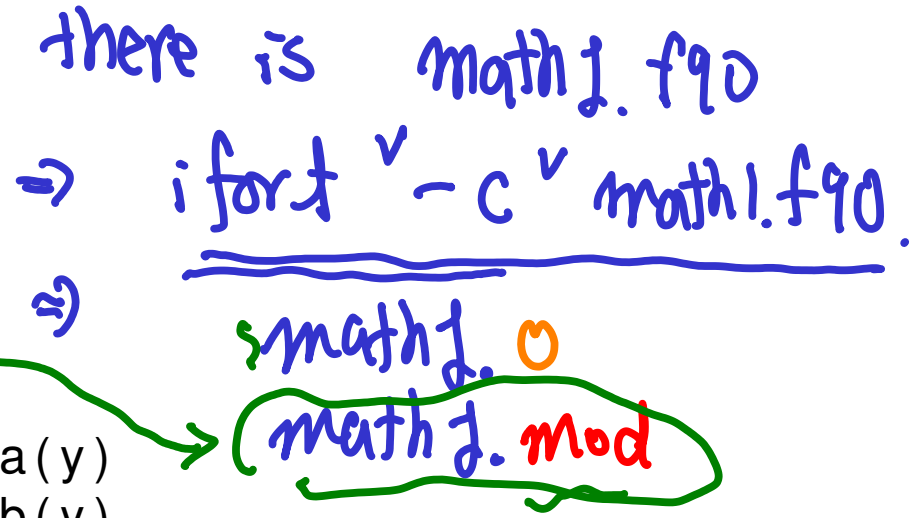
Data type, customized

```
1  module mytype
2     implicit none
3     type person
4        character (len=20) :: pname
5        integer    (kind=8) :: age
6        character (len=14) :: phone(2)
7     end type person
8  end module
9
10 program typetest
11    use mytype
12    implicit none
13    type(person) :: me(2)
14    me(1)%pname="Albert␣S.␣Kim"    ;   me(1)%age=32
15    me(1)%phone(1)="1-808-956-3718"
16    me(1)%phone(2)="1-808-233-9491"
17     write(*,"(A20)") me(1)%pname
18     write(*,"(I2)" ) me(1)%age
19     write(*,"(A14)") me(1)%phone(1)
20     write(*,"(A14)") me(1)%phone(2)
21 end
```

Screen outputs

```
./mytype.x
Albert S. Kim
32
1-808-956-3718
1-808-233-9491
```

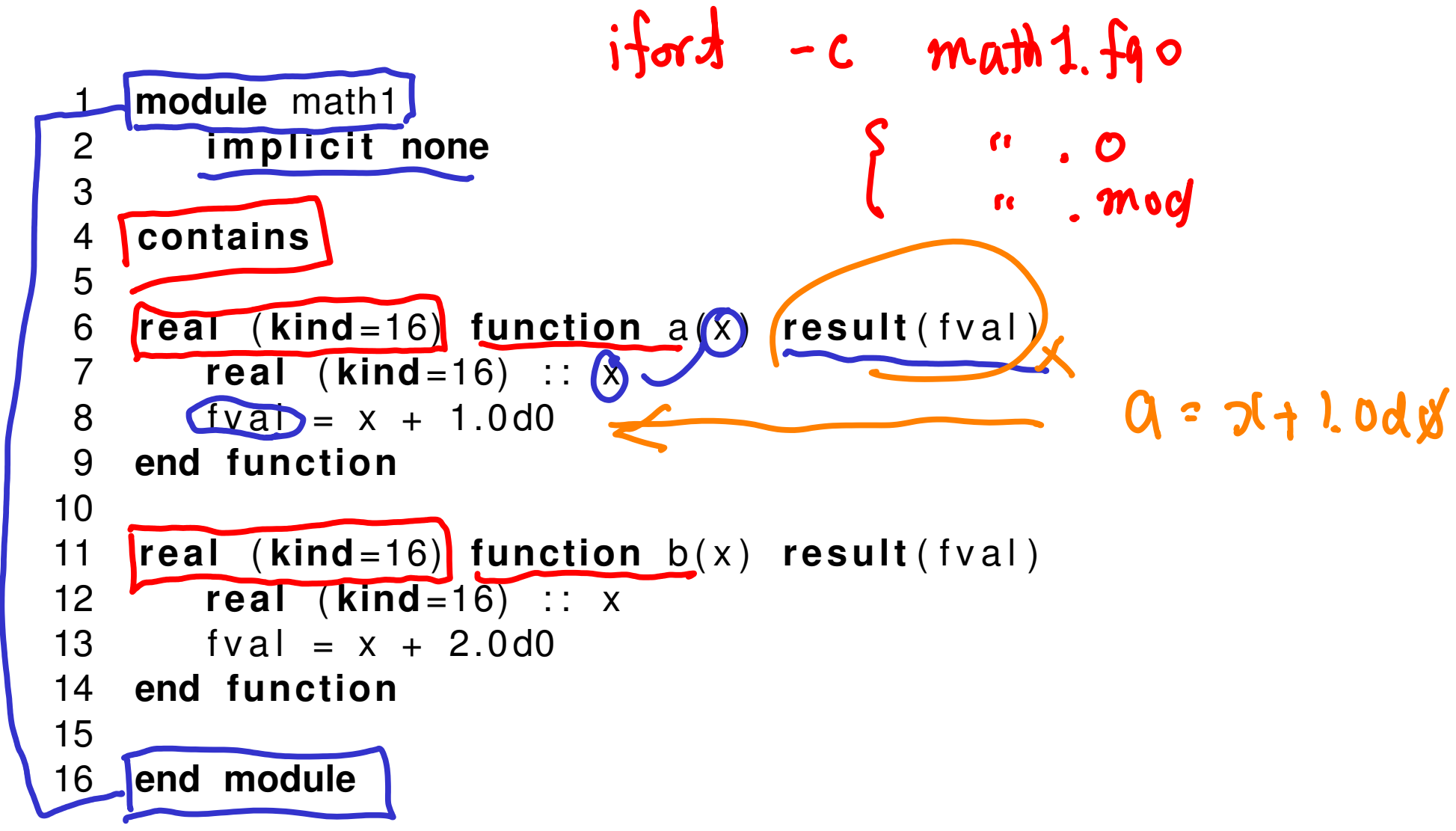
Main program 1 (main1.f90)




```
1   program main1
2   use math1
3   implicit none
4   real (kind=16) :: y = 2.0d0
5   write(*,"(2(2x,e22.15))") y, a(y)
6   write(*,"(2(2x,e22.15))") y, b(y)
7   end program
```

```
./main1.x
0.20000000000000E+01 0.30000000000000E+01
0.20000000000000E+01 0.40000000000000E+01
```

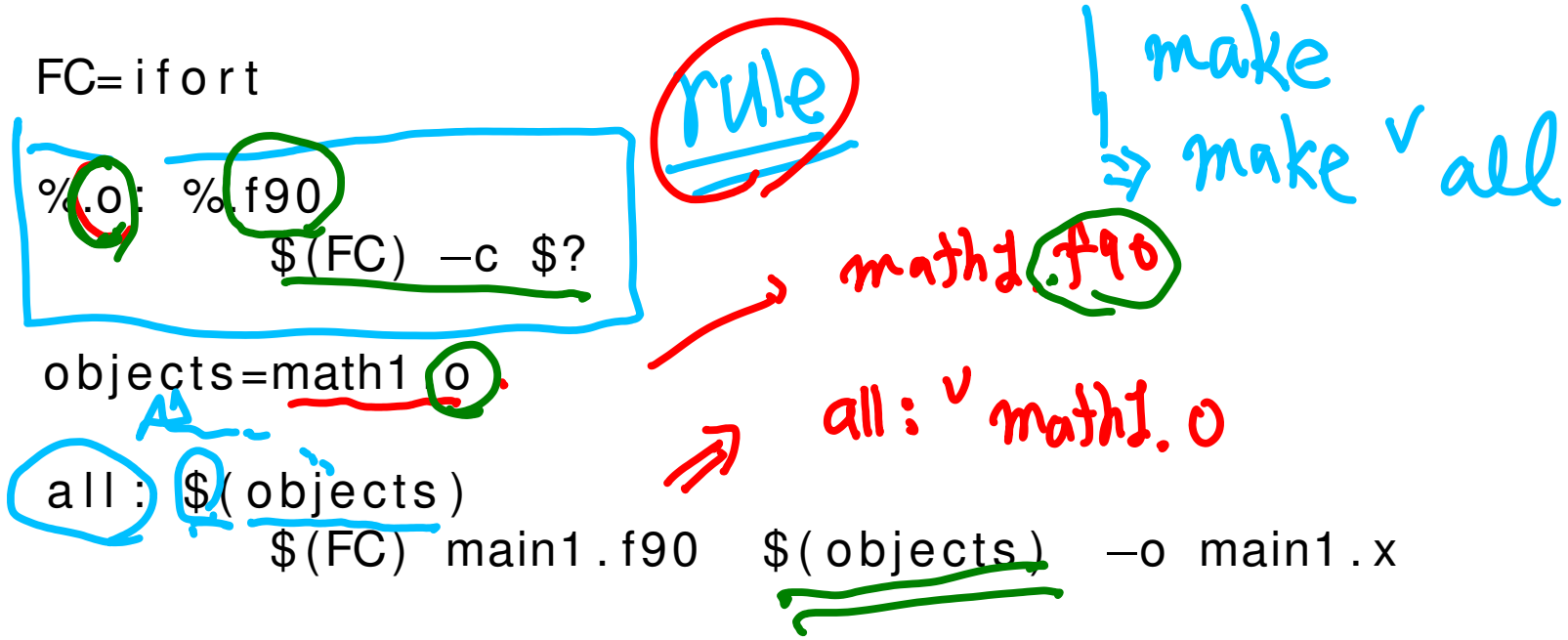
math1 module file (math1.f90)




```
 1  module math1
 2      implicit none
 3
 4  contains
 5
 6  real (kind=16) function a(x) result(fval)
 7      real (kind=16) :: x
 8      fval = x + 1.0d0
 9  end function
10
11  real (kind=16) function b(x) result(fval)
12      real (kind=16) :: x
13      fval = x + 2.0d0
14  end function
15
16  end module
```

Makefile

```
1  FC=ifort
2
3  %.o: %.f90
4          $(FC) -c $?
5
6  objects=math1.o
7
8  all: $(objects)
9          $(FC) main1.f90 $(objects) -o main1.x
10
11 run:
12         ./main1.x
13 clean:
14         rm -f *.o *.x *.mod
```



- “make” command will show (afer “make clean”)

```
$ ifort -c math1.f90
$ ifort main1.f90 math1.o -o main1.x
```

Main program 2 (main2.f90)

```
 1  program main
 2  use math1
 3  use math2
 4  implicit none
 5  real (kind=16) :: y = 2.0d0
 6  write(*,"(2(2x,e22.15))") y, a(y)
 7  write(*,"(2(2x,e22.15))") y, b(y)
 8  write(*,"(2(2x,e22.15))")
 9  write(*,"(10(2x,f16.9))") y, a2(y),  b2(y), b3(y), b4(y)
10  end program
```

./main2.x

0.200000000000000E+01 0.300000000000000E+01

0.200000000000000E+01 0.400000000000000E+01

2.000000000 3.000000000 4.000000000 4.000000000 4.000000000

math2 module file (math2.f90)

```
 1  module math2
 2      implicit none
 3  contains
 4  real (kind=16) function a2(x) result(fval)
 5      real (kind=16) :: x
 6      fval = x + 1.0d0
 7  end function
 8  real (kind=16) function b2(x) result(fval)
 9      real (kind=16) :: x
10      fval = x + 2.0d0
11  end function
12
13  real (kind=16) function b3(x) result(fval)
14      real (kind=16) :: x
15      fval = a2(x) + 1.0d0
16  end function
17  real (kind=16) function b4(x) result(fval)
18      use math1, only : a
19      real (kind=16) :: x
20      fval = a(x) + 1.0d0
21  end function
22  end module
```

// a(x) in math1.f90

// b(x) in math1.f90

⇒ fval = x + 2.0d0

math1 module file (math1.f90)

Again ...

```
 1  module math1
 2      implicit none
 3
 4  contains
 5
 6  real (kind=16) function a(x) result(fval)
 7      real (kind=16) :: x
 8      fval = x + 1.0d0
 9  end function
10
11  real (kind=16) function b(x) result(fval)
12      real (kind=16) :: x
13      fval = x + 2.0d0
14  end function
15
16  end module
```

Makefile

```
1  FC=ifort
2  %.o: %.f90
3          $(FC) -c $?
4  objects=math1.o     math2.o
5
6  all: $(objects)
7          $(FC) main2.f90  $(objects)  -o main2.x
8
9  run:
10          ./main2.x
11 clean:
12         rm -f *.o *.x *.mod
```

- “make” command will show (afer “make clean”)

```
$ ifort␣-c␣math1.f90
$ ifort␣-c␣math2.f90
$ ifort␣main1.f90␣math1.o␣math2.o␣-o␣main1.x
```

Main program 2 (main2.f90)

```
use physics
implicit none
write(*,"('␣Pi␣value␣is␣',1x,F22.15)")   pi
write(*,"('␣Avogadro␣number␣is␣',1x,e22.15)") Na
write(*,"('␣Gas␣constant␣␣is␣',1x,e22.15)")    Rg_J_MolKlvn
write(*,"('␣Boltzmann␣constant␣is␣',1x,e22.15)") KBoltzman
end
```

physics module file (physics.f90) and screen output

```
module physics
implicit none
real (kind=16), parameter :: pi=3.14159265358979323846D0
real (kind=16), parameter :: Na=6.02214129d23 ! per mole
real (kind=16), parameter :: Rg=8.3144621 ! J / mol K
real (kind=16), parameter :: Rg_J_MolKlvn = Rg
real (kind=16), parameter :: Rg_kJ_MolKlvn=8.3144621D-3 ! J/mol K
real (kind=16), parameter :: KBoltzman = 1.3806488d-23 ! Joule/K
end module
```

- Screen output by “make” followed by “make run”

Pi value is 3.141592653589793

Avogadro number is 0.602214129000000E+24

Gas constant in [J/ mol K] is 0.831446170806885E+01

Boltzmann constant is 0.138064880000000E-22

Outline

opi4.f90

```
program main
use omp_lib
double precision  PI25DT
parameter         (PI25DT = 3.14159265358979323846264 3d0)
double precision  pi, h, sum, x, f, a,  time, T1, T2
integer           n,i
f(a) = 4.d0 / (1.d0 + a*a)
n = 1250000000
h = 1.0d0/dble(n)
sum  = 0.0d0
T1 = TIME()
!$omp parallel do private (i,x) reduction (+:sum)
do i = 1, n
    x = h * (dble(i) - 0.5d0)
    sum = sum + f(x)
enddo
pi = h * sum
T2 = TIME()
write(*, *) pi, abs(pi - PI25DT)
stop
end
```

MPI_REDUCE

MPI_SUM

do i = myid+1, n, numprocs

Compile and Run

- Compile

  $ ifort -openmp opi4.f90 -o opi4.x

- Determine the number of threads. Try 1, 2, 4, 6, and 8

  $ export OMP_NUM_THREADS=8

- Execution is identical to serial cases. Measure thr running time.

  $ time ./opi4.x

opi7.f90

```
 program main
 use omp_lib
 double precision  PI25DT
 parameter         (PI25DT = 3.14159265358979323846264 3d0)
 double precision  pi, h, sum, x, f, a, T1, T2, time
 integer n, myid, numthreads,   i
 f(a) = 4.d0 / (1.d0 + a*a)
 n=1250000000  ; h=1.0d0/dble(n) ; sum=0.0d0 ; T1=TIME()
!$OMP PARALLEL private (myid)
 myid       = OMP_GET_THREAD_NUM()
 numthreads = omp_get_num_threads()
 write(*,*) myid, numthreads
!$OMP  DO SCHEDULE(STATIC, numthreads) reduction(+:sum)
 do i = 1, n
    x = h * (dble(i) - 0.5d0)
    sum = sum + f(x)
 enddo
!$OMP end do
!$OMP END PARALLEL
 pi = h * sum ; T2 = TIME()  ; write(*, *) pi, abs(pi - PI25DT)
stop
```

opi3.f90

```
program main
use omp_lib
double precision  PI25DT
parameter        (PI25DT = 3.14159265358979323846264 3d0)
double precision  pi, h, sum, x, f, a, T1, T2, time
integer n, i
f(a) = 4.d0 / (1.d0 + a*a)
n=1250000000       ;  h = 1.0d0/dble(n) ; sum  = 0.0d0
T1 = TIME()
!$omp parallel private (i,x)
!$omp do reduction (+:sum)
do i = 1, n
    x = h * (dble(i) - 0.5d0)
    sum = sum + f(x)
enddo
!$omp end do
!$omp end parallel
pi = h * sum
T2 = TIME()
write(*, *) pi, abs(pi - PI25DT)
stop
```

Discussion

OpenMP

1. is relatively easy to learn in comparison to MPI, but the performance is not straightforward.
2. is better for small shared memory systems when 4-8 times speedup is remarkable.
3. is more sensitively depending on compilers and hardware.
4. does not communicate through network but channels through processors/cores.
5. is not the best for (very) large scale simulation with flexibility.
6. can be used toghether with MPI, called hybrid parallelization.