

CEE 618 Scientific Parallel Computing (Lecture 2): FORTRAN, Library, Make, and PBS

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

- 1 FORTRAN
- 2 Numerical Library
 - Monty Hall Dilemma
 - Wine Selection
- 3 “make” utility

Outline

- 1 FORTRAN
- 2 Numerical Library
 - Monty Hall Dilemma
 - Wine Selection
- 3 “make” utility

FORTRAN

- All the sample codes are located under **/opt/cee618s13/** from which you can copy files.
- I recommend you to generate the same directory structure under your \$HOME directory.

```
$_ mkdir _cee618s13
$_ cd _cee618s13
$_ mkdir _class01
$_ mkdir _class02
```
- Then, store all files of the last class in 'class01' and today's files in 'class02'.

```
$_ cd _class02
$_ cp -r _/opt/cee618s13/class02/* _./
```

Summation code from 1 to 100, $\sum_{i=1}^{100} n$

```
1 program sum100
  implicit none
3 integer :: i, n=100, nsum=0
  do i = 1, n
5     nsum = nsum + i
  enddo
7 print *, nsum
  stop
9 end
```

codes/sum/sum.f90

This code can be modified for your homework to calculate $\sum_{i=1}^{100} n^3$

How to compile and run

- `$ ifort sum.f90 -o sum.x ↵`
- `$ sum.x ↵`

5050

Outline

1 FORTRAN

2 Numerical Library

- Monty Hall Dilemma
- Wine Selection

3 “make” utility

Monty Hall Paradox

Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?



Search "The Monty Hall Problem" in YouTube:

<http://www.youtube.com/watch?v=mhlc7peG1Gg>

Possible Answers: What is your own answer?

- 1 Switch
- 2 Stay
- 3 No difference

Correct Answer: Switch

- 1 Reference: Chapter 1. "*Bias Monte Carlo Methods in Environmental Engineering*" in "Applications of Monte Carlo Methods in Biology, Medicine and Other Fields of Science" edited by Charles J. Mode, ISBN 978-953-307-427-6. Freely downloadable at <http://albertsk.org/publications/>
- 2 Test an example case.

Selecting Good Wines



- A wine box has 12 bottles, 3 of which contain spoiled wine. A sample of 4 bottles is randomly selected from the box. Find the probability that you don't selected any spoiled wine.

(Answer)

$$p = \frac{C_0^3 C_4^9}{C_4^{12}} = \frac{1 (126)}{495} = 0.254545 \quad (1)$$

where

$$C_k^N = \frac{N!}{k! (N - k)!} \quad (2)$$

The number of ways of picking k unordered outcomes from N possibilities. Read " N choose k ."

Program

There are is a main program file, wine.f90, and two additional files, selectn.f90, and spoiled.f90, which of each contains one subroutine.

- nran: the number of wine bottles randomly selected, e.g., 4 out of 12.
- nunspoiled: the number of events that spoiled wine bottles are not selected.
- ntotal: the total number of computer experiments.
- ID: the ID array of randomly selected bottles with “nran” elements. For example, if four bottles [2, 5, 7, and 10] contain spoiled wine, then ID(1)=2, ID(2)=5, ID(3)=7, and ID(4)=10.
- wine: logical data
 - 1 If wine is “.true.”, then none of “nran” bottles contains spoiled wine.
 - 2 If wine is “.false.”, then at least one of “nran” bottles contains spoiled wine.
- Probability = the number of selecting “nran” unspoiled wine bottles divided by the total number of trial, i.e., nunspoiled / ntotal.

wine.f90

```
1 program wine_select
  implicit none
3 integer          :: i, j, k, nunspoiled=0
  integer, parameter :: nbottle=12, nspoiled=3, nran=4, ntotal
    =1000000
5 integer          :: ID(nran)
  real             :: probability
7 logical          :: wine

9 do i = 1, ntotal
  call selectn(nbottle, nran, ID)
11 call spoiled(nspoiled, nran, ID, wine)
  if (wine == .true.) nunspoiled = nunspoiled + 1
13 enddo
  probability = DBLE(nunspoiled) / DBLE(ntotal)
15 write(*, "('Probability is', 1X, F9.6)") probability

17 stop
end
```

selectn.f90

```
subroutine selectn( nbottle , nran , ID )
2 implicit none
integer :: nbottle , nran , ID( nran ) , i , j , jproduct
4 real    :: X
logical  :: overlap
6 call random_number(X)
ID(1) = int( dble(nbottle) * X + 1.0 )
8 do i = 2, nran
    overlap=.true.
10    do while (overlap==.true.)
        call random_number(X)
12        ID(i) = int( dble(nbottle) * X + 1.0 ) ; jproduct = 1
        do j = 1, i-1
14            jproduct = jproduct * (ID(j)-ID(i) )
        enddo
16        if(jproduct/=0) overlap =.false.
    enddo
18 enddo
return
20 end
```

spoiled.f90

```
1 subroutine spoiled(nspoiled, nran, ID, wine)
2 implicit none
3 integer :: i
4 integer :: nspoiled, nran, ID(nran)
5 logical :: wine
6 wine = .true.
7 do i = 1, nran
8     if (ID(i) <= nspoiled) then
9         wine = .false.
10        exit
11    endif
12 enddo
13 return
14 end
```

codes/wine/spoiled.f90

Program and compilation

- Program structure
 - 1 The main program wine_select stored in 'wine.f90' file calls two subroutines.
 - 2 These are selectn (nbottle,nran, ID) and spoiled (nspoiled, nran, ID, wine), stored in files 'selectn.f90' and 'spoiled.f90', respectively.
 - 3 Subroutine selectn (nran, ID) calls intrinsic FORTRAN 90 subroutine random_number (X), where $0 < X < 1.0$ is a random number.
 - 4 In total, 3 files = 1 main program + 2 subroutine files.
- Simple compilation
 - 1 `ifort_□ *.f90`
 - 2 `ifort_□ wine.f90_□ selectn.f90_□ spoiled.f90`
 - 3 `ifort_□ wine.f90_□ selectn.f90_□ spoiled.f90_□ -o_□ wine.x`
- In compilation 1 and 2, "a.out" is created as an executable file. And, in compilation 3, "wine.x" as specified.

Advanced compilation using object files

- Object file
 - ① Object file = a file of **sequences of computer instructions**, which may be *directly executed by a computer's CPU*.
 - ② The following command will generate an object file **selectn.o** from a source file **selectn.f90**.
`$ ifort -c selectn.f90`
 - ③ Why and when do we generate object files?
Subroutines/functions are already made and ready to use.
- Method 1: compile subroutine files individually and use them.
 - ① `ifort -c selectn.f90`
 - ② `ifort -c spoiled.f90`
 - ③ `ifort wine.f90 selectn.o spoiled.o -o wine.x`
- Method 2: all the subroutine files can be compiled together.
 - ① `ifort -c selectn.f90 spoiled.f90`
 - ② `ifort wine.f90 selectn.o spoiled.o -o wine.x`
- All equivalent to: `ifort wine.f90 selectn.f90 spoiled.f90 -o wine.x`

What is “library”?

- A collection of object files ready to use.
- How to generate a library called “libwine.a” using selectn.f90 and spoiled.f:
 - 1 `ifort -c selectn.f90 spoiled.f`
 - Generates `selectn.o` and `spoiled.o` object files.
 - 2 `ar -r -v libwine.a selectn.o spoiled.o`
 - “ar” indicates **archive** with options “-r” to **replace** (also create if not present) and “-v” for **verbose**. In other words, the two subroutine files are contained in a newly generated library file “libwine.a”. Note that a library file should have extension “.a”.
 - 3 `ifort wine.f90 libwine.a -o wine.x`
 - The main code compilation includes the library file “libwine.a”, which can contain other subroutines (not used in wine.f90) as well.
 - In the above command, “libwine.a” replaces the two object files.
 - 4 (*alternatively but important*) `ifort wine.f90 -L./ -lwine -o wine.x`
 - “-L” indicates that what follows is a library path, which is in this case the current directory “./”. “-l” means library file of extension “.a” with partial name “wine” after “lib”, i.e., “libwine.a”

Outline

1 FORTRAN

- 2 Numerical Library
- Monty Hall Dilemma
 - Wine Selection

3 “make” utility

“Make utility”

- 1 make - GNU make utility to maintain groups of programs
- 2 The purpose of the make utility is to determine automatically which pieces of a large program need to be recompiled, and issue the commands to recompile them.
- 3 To prepare to use make, you must write a file called the “makefile” or “Makefile” that describes the relationships among files in your program, and states the commands for updating each file. In a program, typically the executable file is updated from object files, which are in turn made by compiling source files.
- 4 Once a suitable makefile exists, each time you change some source files, this simple shell command:

```
$ make
```

This will update objects files whose source codes are younger.

- 5 If you prepared a Makefile as “Makefile.2”, use “-f” option:

```
$ make -f Makefile.2
```

Makefile

```

all:
2   ifort wine.f90 selectn.f90 spoiled.f90 -o wine.x
run:
4   ./wine.x
edit:
6   vim wine.f90

```

codes/wine/Makefile

- To compile codes using make utility: `make` ↵
("make" utility will read the default file "Makefile".)
- Equivalent to: `make` ↵ `all` ↵
- To execute "wine.x": `make` ↵ `run` ↵
- "all" and "run" are called **targets**.
- The line below each target should start with **Tab**, followed by **command**.
- Target name "all" can be changed to one corresponding a project such as "WineProject" (case-sensitive).

Makefile.1

```
all:
2   ifort wine.f90 selectn.f90 spoiled.f90 -o wine.x
run:
4   ./wine.x
```

codes/wine/Makefile.1

- The same content is stored in "Makefile.1"
- Equivalent to: `$ make -f Makefile.1 all ↵`
- To execute "wine.x": `$ make -f Makefile.1 run ↵`

Makefile.2

```
all: selectn.o spoiled.o
    ifort wine.f90 selectn.o spoiled.o -o wine.x
spoiled.o:
    ifort -c spoiled.f90
selectn.o:
    ifort -c selectn.f90
run:
    ./wine.x
```

codes/wine/Makefile.2

- Again
 - "\$ make -f Makefile.2 ↵" and "\$ make -f Makefile.2 all ↵" are identical.
- Dependency for target: when a target is made, dependencies will be made **first** (if not made).

Makefile.3

```
1 all: lib
   ifort wine.f90 -L./ -lwine -o wine.x
3 # ifort wine.f90 libwine.a -o wine.x
lib: selectn.o spoiled.o
5 ar -r -v libwine.a selectn.o spoiled.o
spoiled.o:
7 ifort -c spoiled.f90
selectn.o:
9 ifort -c selectn.f90
run:
11 ./wine.x
clean:
13 rm -f selectn.o spoiled.o libwine.a wine.x
```

codes/wine/Makefile.3

- "libwine.a" library is made using dependencies.
- "clean" **removes** all generated files, except source and Makefile(s).