Lecture 1: Introduction to Round-off Error□□

Round-off error plays an important role on the stability and accuracy of a computer simulation. The goal of this lecture is to show where the round-off errors come from and how to minimize the impact of the round-off errors in the numerical simulation.

In order to understand the source of the round-off errors, we need to know the computer architecture as well as the computer representations of the integer numbers and the real numbers.

1.1. Review of Word, Byte, and Bit in the Computer Architecture

In the modern computer processor, a byte consists of 8 bits. A word consists of 4 bytes for 32 bits computer or 8 bytes for 64 bits computer. Thus, the size of word is 32 bits for 32-bit computer, but 64 bits for 64-bit computer.

Exercise 1.1.

To learn more about the historical development of the computer architecture, please read the information on "word in computer architecture" in Wikipedia, http://en.wikipedia.org/wiki/Word_(computer_architecture)

From Exercise 1.1, it can be seen that the most popular sizes of word found in different processors are 64, 32, 16, 8, and 4 bits. But one can also found size of word to be 60, 50, 48, 40, 39, 36, 34, 27, 26, 25, 22, 18, 15, 12, or 9 bits. The size of byte is equal to the size of character. For most processor, the size of byte is 8 bits. But there were also processor with size of byte equal to 5, 6, or 9 bits.

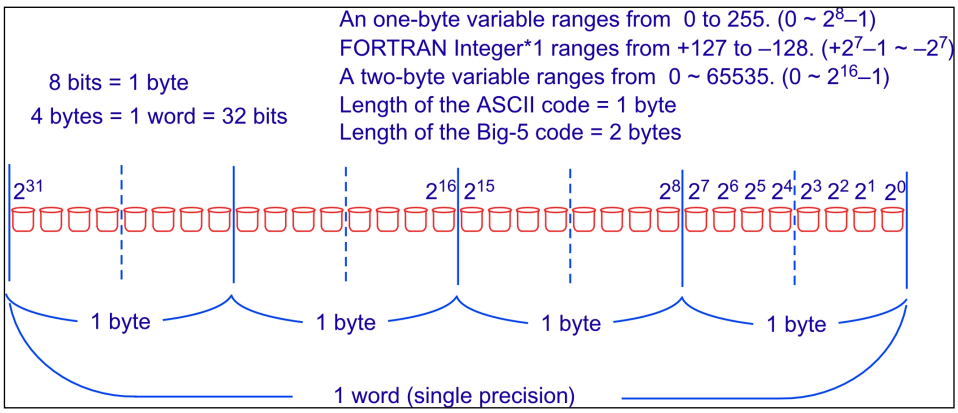

Figure 1. The structure of word, bytes, and bits in a 32-bit computer

1.2. Computer Representations of Integer Numbers

Figure 1 shows the relationship among a single precision word, byte, bit in a 32-bit computer. For the modern 32-bit computer, a single precision word consists of 4 bytes. A byte consists of 8 bits. Thus, a single precision word consists of 32 bits. From Figure 1, we can also conclude that, for the 32-bit computer, the maximum integer register is 2^{32}=4294967296 ~ 4G. But the range of integer depends on how the processor treats the negative integer. For the modern computer processor, the computer representation of integer is given in the following way:

- When the first bit is 0, the integer is positive. The absolute value of the positive integer is determined by the binary representation of the rest 31 bits.
- When the first bit is 1, the integer is negative. The absolute value of the negative integer is determined by the binary representation of the complement of the rest 31 bits plus 1.

Q: What is $(10011000)_2$?
The first bit is 1. It means it is a negative integer. Changing the rest bits from 0011000 to 1100111, it yields $(1100111)_2$= 64 + 32 + 0 + 0 + 4 + 2 + 1 = 103.
The absolute value of the negative integer is 103 + 1 = 104. Thus, $(10011000)_2$= – 104
(When the author was a graduate student in NCU, the university has purchased a 60-bits CDC-Cyber computer. In order to speed up the calculation, this computer taking the complement of the rest 59 bits without adding 1 when it evaluates the negative integers. As a result, both $(000\ldots000)_2$ and $(111\ldots111)_2$ equal to 0 in the 60-bits CDC-Cyber computer.)

Exercise 1.2.

Read the information on "signed number representations" in Wikipedia,
http://en.wikipedia.org/wiki/Signed_number_representations

Exercise 1.3.

Read the information on "integer in computer science" in Wikipedia,
http://en.wikipedia.org/wiki/Integer_(computer_science)

1.3. Computer Representations of Integers and Characters

The integers and characters are connected through a given code table. Examples of such code include the ASCII code for English and the Big-5 code for Traditional Chinese. One may find intrinsic functions to make converting between the integers and ASCII code. Or, one can use "A format" in FORTRAN language to convert integers and "characteristics" (or sometimes called "strings"). However, it is even more important to know that we can output our integer or floating-point data in "A format" as binary data to save a lot of disk space.

Note that after year 2000, the formatted binary data has become a machine-independent data structure. Thus, most of the computer graphic software, such as IDL or MATLAB, has the ability to read the binary data structure. But the unformatted binary data structure is remained to be machine-dependent data structure. Thus, one has to plot the data in the same machine if the output data file is an unformatted binary data.

Exercise 1.4

Write a test program to find out the binary structures of the ASCII code, the integers, and the floating numbers used in your computer. (e.g., Write 31~127 in A1 format to a file and find out what you can see in that file.)

The following are two FORTRAN programs for ASCII code and Chinese Big-5 code

```
C==GETASCII.f===========
C  This program shows the binary structures of the ASCII code
      Program GETASCII
      DO I=31, 127
      WRITE(3,20) I, I
      ENDDO
   20 FORMAT(1X,I3,1X,A1)
      STOP
      END
```

```
C==GETBIG5.f============
C  This program outputs the Chinese characters in Big-5 code.
C  To view the Chinese chsaracters, you can open the output file by a web browser
C     and choose viewing text by Big-5 encoding.  Then, you can make
C     a copy of the Chinese characteristics and past them to a regular word document.
C
```

```
      PROGRAM GETBIG5
C HIGH: A1-F9 (161-249)
C LOW:  40-7E (64-126), A1-FE (161-254)
      INTEGER*1 IA(20000),IB(20000)
      JJ=0
      DO I=161,249
        DO II=64,126
        JJ=JJ+1
        IB(JJ)=II
        IA(JJ)=I
        ENDDO
        DO II=161,254
        JJ=JJ+1
        IB(JJ)=II
        IA(JJ)=I
        ENDDO
      ENDDO
      JJ0=JJ
      WRITE(11,1) (IA(K),IB(K),K=1,JJ0)
    1 FORMAT(100A1)
      STOP
      END
```

Additional example of FORTRAN program to determine the ASCII code of a given character and vice versa.

```
C==ASCII_TEST.f=======================
C This program determines the ASCII code of a given character and vice versa
      program ascii_test
      character*1 a
      byte i
      id=1 !id can be any integer between 1 and 99 except 5 and 6
           !id=5 is reserved for the system_input, such as the terminal
           !id=6 is reserved for the system_output, such as the terminal
   10 continue
      print *,'enter one character'
      read(5,*) a
      write(id,1)a
    1 format(A1)
      rewind id
      read(id,1)I
      print *,'I=',I
      rewind id
      write(id,2)a
    2 format(A2)
      rewind id
      read(id,2)I
```

```
      print *,'I=',I
C
      print *,'enter an integer, or enter 0 to stop'
      read(5,*) I
      if(i.eq.0) go to 99
      rewind id
      write(id,1)I
      rewind id
      read(id,1)a
      print *,'a=',a
      rewind id
      write(id,2)I
      rewind id
      read(id,2)a
      print *,'a=',a
      go to 10
   99 continue
      stop
      end
```

1.4. Computer Representations of Integers and Real Numbers

Table 1.1 shows the lower and upper limits of the integers and byte(s) at different length.

Table 1.1. The lower and upper limits of the integers and byte(s) at different length

Fortran Data Type	lower limit of the data	upper limit of the data
Byte	0	+ 255 (= 2^8 – 1)
Integer*1	– 128 (= – 2^7)	+ 127 (= 2^7 – 1)
Bytes	0	+ 65535 (= 2^{16} – 1)
Integer*2	–32768 (= – 2^{15})	+ 32767 (= 2^{15} – 1)
Integer*4	– 2147483648 (= – 2^{31})	~ 2147483647 (= 2^{31} – 1)

Exercise 1.5.

To learn more about the historical development of the floating point in the computer architecture, please read the information on “floating point” on Wikipedia, http://en.wikipedia.org/wiki/Floating_point

Exercise 1.6

Write a program to verify the results shown in Table 1.1.

Table 1.2 shows the computer representation of floating point at different length, which is modified from the webpage discussed in Exercise 1.5. The significant digits listed in Table 1.2 will give raise to round-off error. This is the reason why we must use double precision in our numerical simulation.

Note that there is no round-off error in the integer expression and calculation. But the maximum and minimum in the integer expression is much less than the extrema of the floating-point number at the same length. Both real number and complex number are floating point numbers with finite significant digits. Since a complex number consists of two real numbers, which represent the real part and the imaginary part of the complex number, the length of a complex number is twice of that of a real number.

Table 1.2. The computer representation of the floating points at different length

Type	Sign	Exponent	Significand	Total bits	Exponent upper limit	significant digits
Half (IEEE 754-2008)	1	5(=1+4)	10	16	15(=2^4–1)	~3.3
Single	1	8(=1+7)	23	32	127(=2^7–1)	~7.2
Double	1	11(=1+10)	52	64	1023(=2^{10}–1)	~15.9
Double extended (80-bit)	1	15(=1+14)	64	80	16383(=2^{14}–1)	~19.2
Quad	1	15(=1+14)	112	128	16383(=2^{14}–1)	~34.0

Exercise 1.7

(a) Write a program to verify the last column shown in Table 1.2.

(b) Write a program to check the value in the first three columns shown in Table 1.2 in your computer.

List below is an example of Fortran program, which determines the extrema of the real number and the integer number that can be resolved by the current computing system.

```
C==MAXIMUM_TEST.f====================================================
      PROGRAM MAXIMUM_TEST
      REAL*8 AA
    1 CONTINUE
      PRINT *,'ENTER AA, A, I'
      READ(5,*) AA, A, I
      B=EXP(A)
      IF(I.EQ.0) GO TO 99
```

```
      PRINT *, 'AA, A, EXP(A), I ='
      PRINT *, AA, A, B, I
      GO TO 1
   99 CONTINUE
      STOP
      END
```

1.5. How to Determine the Relative Error of a Floating Number

If U is the relative error of 1, then an iteration scheme is convergent when $\left|{}^{k+1}y^{n+1} - {}^{k}y^{n+1}\right| < U\left|{}^{k}y^{n+1}\right|$.

where ${}^{k}y^{n+1}$ is the k th iteration result of y^{n+1}.

The relative error U can be obtained from the program `GETU.f` as given below. (e.g., Shampine and Gordon, 1975). The relative error of a number A is U*|A|. The relative error is a machine-dependent error before year 2000. The reason that different floating-point processor has different relative error can be understand by the historical review given in Exercise 1.5.

```
C== GETU.f ==========
C  This subroutine determines machine-dependent relative error
C   relative to 1.
      Subroutine GETU(U)
      Implicit double precision (a-h,o-z)
      A1=1.d0    !for double precision
      AH=0.5d0    !for double precision
C      A1=1.     !for single precision program
C      AH=0.5     !for single precision program
      U=A1
      UU=U
    1 CONTINUE
      UU=UU*AH
      UT=U+UU
      IF(UT.GT.U) GO TO 1
      U=UU*2
      RETURN
      END
```

1.6. How to Minimized the Numerical Errors due to Round-off Error

We can minimize the numerical errors due to round-off errors by the following way.

If $|A-B| <$ (the relative error of A and B), set $A-B=0$ (Tsai et al., 2009)

Examples of the function programs and main program that can demonstrate the round-off errors in the derivatives of $\tanh(x)$ are given below.

```
C==DIFAB.f=============
C This function determines A-B with minimized round-off error
      FUNCTION DIFAB(A,B)
      Implicit double precision (a-h,o-z)
      common RELERR
      TEMP=A-B
      IF(DABS(TEMP).LT.MAX(DABS(A),DABS(B))*RELERR) TEMP=0
      DIFAB=TEMP
      RETURN
      END
```

```
C==ADDAB.f========
C This function determines A+B with minimized round-off error
      FUNCTION ADDAB(A,B)
      Implicit double precision (a-h,o-z)
      common RELERR
      TEMP=A+B
      IF(DABS(TEMP).LT.MAX(DABS(A),DABS(B))*RELERR) TEMP=0
      ADDAB=TEMP
      RETURN
      END
```

```
C==test_DIFAB.f========
C This program determines the derivatives of the tanh(x)
      program test_DIFAB
      Implicit double precision (a-h,o-z)
      common RELERR
      call DGETU(RELERR)
      RELERR=RELERR*10
      A01=0.1d0
      A0H=A01/2.d0
      FL=1  ! exp(0)=1
      DO I=1,180
      X=I*A01
      temp=dexp(x)
```

```
      temp1=1.d0/temp
      FR=(temp-temp1)/(temp+temp1)
      PF1=(FR-FL)/A01
      FR=DIFAB(temp,temp1)/ADDAB(temp,temp1)
      PF2=DIFAB(FR,FL)/A01
      IF(I.GT.160) THEN
      WRITE(2,*), 'X1, PF1, PF2='
      WRITE(2,*), X1, PF1, PF2
      ENDIF
      FL=FR
      ENDDO
      STOP
      END
C====================================
      INCLUDE 'DIFAB.f'
      INCLUDE 'ADDAB.f'
      INCLUDE 'GETU.f'
```

1.7. Summary

Computer simulation is a special type of numerical method, which means to solve a system of ordinary differential equations (ODEs) or partial differential equations (PDEs) with a time-derivative term presentted in each of the ODEs or PDEs. Simulation results can provide information on how the given system will evolve with time and why it evolves in such a way. The simulation results will depend on the governing equations, the simulation scheme, the grid size, the size of the simulation domain, the time steps, the initial conditions, and the boundary conditions. Since most of the numerical simulations are time-consuming and memory-consuming, we need to do our best to save the real time, and find a balance between saving the CPU time and saving the memory. For better diagnostics of the simulation results, we need a lot of storage space to save the simulation results. Thus, we need to do our best to save the disk space.

How to Save CPU Time:

- Do not repeat complicate calculations, such as $\sin x$, $\cos x$, $\tan x$, $\mathrm{ATAN2}(x,y)$, $\exp x$, and $\log x$. Set up a table (array) to keep the results of the repeat calculation.
- Use x^{iy} instead of x^{y}, when $y = iy$ is indeed an integer.

How to Save the Real Execution Time:

Reducing the frequency of I/O can greatly reduce the real execution time.

- Reduce the number of times in I/O by increasing the record length of the block size.
- Using internal loop instead of external loop can greatly reduce the frequency of I/O.

How to Save the Disk Space:

- Saving the simulation results in a binary format (A format or unformatted form) instead of ASCII format can greatly reduce the file size and save the disk space.

The following examples are FORTRAN programs in which the first one will take much longer time to complete the execution in comparing with the second program.

```
C==TEST_IO_SLOW.f=============
C THIS PROGRAM SHOWS A BAD EXAMPLE OF I/O
      PROGRAM TEST_IO_SLOW
      PARAMETER(NCX=1000000)
      DO I=1,NCX
      ARRAY=I*0.1D0
      WRITE(8,8) I, ARRAY
    8 FORMAT(1X, I6, 1X, F15.7)
      ENDDO
      STOP
      END
```

```
C==TEST_IO_FAST.f=============
C THIS PROGRAM SHOWS A GOOD EXAMPLE OF I/O
      PROGRAM TEST_IO_SLOW
      PARAMETER(NCX=1000000)
      DIMENSION ARRAY(NCX)
      DO I=1,NCX
      ARRAY(I)=I*0.1
      ENDDO
      WRITE(9,8) (I, ARRAY(I), I=1,NCX)
    8 FORMAT(200A4)
      STOP
      END
```

Note that one can use the following command to find the real execution time in a Linux operating system (OS).

```
time ./a.out
```

where `a.out` is the execution file. An example of execution results is given below.

```
C==Real_CPU_Time.txt=============
Real Execution time and CPU time

$ gfortran TEST_IO_FAST.f
$ time ./a.out

real 0m0.197s
user 0m0.119s
sys  0m0.017s

$ gfortran TEST_IO_SLOW.f
$ time ./a.out

real 0m1.335s
user 0m1.257s
sys  0m0.037s

$ ls -l fort.*
-rw-r--r--  1 lyuling-hsiao  staff  24000000 Mar 25 19:52 fort.8
-rw-r--r--  1 lyuling-hsiao  staff   8010000 Mar 25 19:52 fort.9
```

References

Shampine, L. F., and M. K. Gordon (1975), *Computer Solution of Ordinary Differential Equation: the Initial Value Problem,* W. H. Freeman and Company, San Francisco.

Tsai, T. C., L. H. Lyu, J. K. Chao, M. Q. Chen, and W. H. Tsai (2009), A theoretical and simulation study of the contact discontinuities based on a Vlasov simulation code, J. Geophys. Res., 114, A12103, doi:10.1029/2009JA014121.