**과학계산을 위한 컴퓨터활용**

디렉토리 찾기

$ find . –name ‘comm\*’ -print 이런 형태로 작성

$ Find / -name ‘gfortran\*’ –print gfortran 포함한 이름 찾기. /는 루트 디렉토리

$ Find / -name ‘gfortran\*’ –print >gfortout gfortout에 결과물을 저장한다.

$ which gfortran 파일명을 확실히 알면 위치 알려줌

$ find /usr/local –name ‘gfortran’ path 경로 지정은 이렇게 문법 지키면 된다.

$ locate ‘gfortran’ locate도 가능. find보다 더 관련된 것을 찾아준다.

파일 이름 바꾸기

$ mv command.odt com.odt command.odt 를 com.odt로 바꾼다.

예비 복사하기

$ cp com.odt comorigin.odt 현 위치에 복사한다.

삭제 하기

$rm –r dir이름 이름이 해당된 디렉토리를 통째로 지움

현 서버 유저 파악

$ top mem 등 정보 확인 가능

$ w 지금 누가 들어와있는지 확인.log history

FORTRAN Lecture Note

Computer Representation of Numbers and Computer Arithmetic

4.1 Binary Numbers

4.2 Memory

0 1 각각을 bit이라 한다.

1 byte = 8 bit

Double real은 8 byte=64bit 이다.

8 bit number는 다음과 같이 저장된다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0+ | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

4.3 Representation of Signed Integers

Fortran default는 4byte=32bit 32개 방이 기본 데이터형.

|  |  |  |
| --- | --- | --- |
| 0(연산 결정) | 8 exponent | 23(mantissa 유효숫자 결정) |

Real(4) :: single precision

만일 표현할 수 있는 숫자의 크기를 넘어가면 overflow 등장

방 32개는 소수점에서 9번째 자리부터는 거의 무시해버린다.

Real(8) 로 두면, 64개의 방으로, 소수점에서 17번째 자리에서는 거의 junk.(무시) 수치해석으로 구한 방정식의 정확도를 판단할 때 매우 중요.

이제 signed integer를 컴퓨터로 어떻게 표현하는지 알아보자.

-3은 [1]11, -2는 [1]10. -1은 [1]01. -0은 [1]00. +0은 [0]00. +1은 [0]01, +2는 [0]10 …

4.3.1 Integers in Memory

-one byte 당 127 ~ 126 혹은 0~255 숫자 표현 가능

다음을 컴파일해보자

Program test\_int

IMPLICIT NONE

INTEGER::m,i

m=2147483645

do i=1,10

print\*, 'i=',i,'. m+1=',m+i

END DO

END PROGRAM test\_int

이를 실행하면 overflow를 얻는다. Default가 4byte로 되어있기 때문이다. 다음과 같이 수정한 후 실행해보자.

Program test\_int

IMPLICIT NONE

INTEGER(8)::m,i

m=2147483645

do i=1,10

print\*, 'i=',i,'. m+1=',m+i

END DO

END PROGRAM test\_int

제대로 나온다.

두 번째 자리를 9로 바꾸면, 정수가 너무 크다면서 컴파일조차 되지 않는다. 따라서, 해당하는 데이터형을 알맞게 사용하는 것이 중요.

4.4 Floating-Point Numbers

Sign + Mantissa + Exponent 구성을 mantissa is normalized 되었다고 한다.

107.625 = +1 \* 1.07625 \* 10^2

만일 6 decimal digits per FP number에 대하여 다음과 같이 할당된다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sign | Mantissa | Mantissa | Exponent | Exponent | Exponent |

아까의 예시로 들면 다음과 같다.

|  |  |  |
| --- | --- | --- |
| + | 107 | +2 |

이진법의 예시로 다음을 생각하자.

1101011.101=+1 \* 1.101011101 \* 2^6

만일 6 binary digits이 mantissa에 가능하고, 4 binary digits이 exponent에 가능하다면, 이는 다음과 같이 표현된다.

|  |  |  |
| --- | --- | --- |
| +1 | 110101 | 0110 |

방금의 예시에서 오류가 난 이유는, mantissa 영역에 숫자가 잘려서 들어가기 때문이다. 컴퓨터가 화면에 보여줄 때에는 junk를 붙여서 알려주기 때문에 overflow.

4.5 IEEE standard

Single double의 sign mantissa exponent 자리차지 양을 정한 규격. Double이면 mantissa가 52까지 가능한데, 대략 17자리이다.

|  |  |  |
| --- | --- | --- |
| IEEE Format | Machine precision(e) | No. decimal digits |
| Single prec. | 2^-23 ~ 1.2\*10^-7 | 7 |
| Double prec. | 2^-52 ~ 1.1\*10^-16 | 16 |
| Extended prec. | 2^-63 ~ 1.1\*10^-19 | 19 |

수가 너무 작아서 표현 못해 나오는 error는 Underflow 라고 한다.

Code Performance

1) Using a fast algorithm-->Instead of using DFT(O(n^2)), use FFT(O(nlogn))

2) Using unformatted I/O

3) Using higher optimizations levels (Use -OLevel)

4) Performing "aggressive" manual optimization

5) Writing in assembly language

1. time measurement

-Use internal function calls

-system\_clock(int count, int count\_rate, int count\_max): measures wall time

-cpu\_time(real time):measures only cpu time

2. memory optimization

-fully utilizes cache memory -need to know hardware info and assembly language

(-->getconf LEVEL1\_DCACHE\_LINESIZE)

(-->cat /sys/devices/system/cpu/cpu0/cache/index0/coherency\_line\_size)

-memory access pattern-temporal, spatial locality

-Loop unrolling-decrease loop overhead, increase cache hit

(-->possible on the compiler option)

3. fortran compiler optimization

-gfortran -O0 (no optimization)

-gfortran -O2 (optimize except loop unrolling and function inlining)

-gfortran -O3 (-O2 + function inlining + registeries optimization)