UNIVERSITÀ DEGLI STUDI DI PADOVA

Giacomo Gasparotto

Assignment 1

Quantum Information and Computing

A.Y. 25/26

11November 2025

Integer overflow

Sum **2000000** and **1** in two settings:

int16: -31616 + 1 = -31615

Why?

- Let's look at the two-complement:
  $(2000000)_{10} = (111\ 110\ 100\ 001\ 001\ 000\ 000)_2$

- If we store the number in int16 it takes the first 16 most significant digits:
  $(1110\ 0010\ 0100\ 0000)_2 = (-31616)_{10}$

Leading to an error of overflow!

int32: 2000000 + 1 = 2000001

Enough space to store the result.

```
Sum of 2000000 and 1 using 16-bit integers:
-31616 + 1 = -31615
Max interval for 16-bit: [-32768, 32767]

Sum of 2000000 and 1 using 32-bit integers:
    2000000 + 1 = 2000001
Max interval for 32-bit: [-2147483648, 2147483647]
```

UNIVERSITÀ DEGLI STUDI DI PADOVA

Floating point rounding error

Sum two large real numbers in two settings:

real32 – single precision

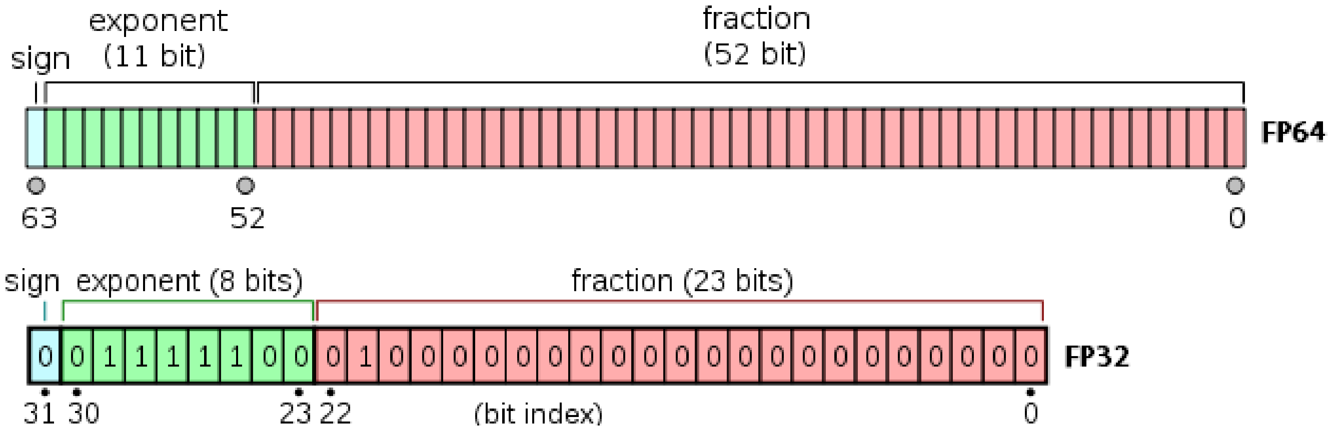
- $\pi \cdot 10^{32}$: 3.14159278E+32
- $\sqrt{2} \cdot 10^{21}$: 1.41421360E+21
- **sum**: 3.14159278E+32

The second number is “lost” due to **rounding** because of 10 order of magnitudes of difference.

real64 – double precision

- $\pi \cdot 10^{32}$: 3.141592653589793 3E+032
- $\sqrt{2} \cdot 10^{21}$: 1.4142135623730950E+021
- **sum**: 3.1415926536039354E+03

The mantissa is large enough to store the entire sum.




```
Single precision real numbers to sum:
  3.14159278E+32
  1.41421360E+21
Single precision sum:    3.14159278E+32

Double precision real numbers to sum:
  3.1415926535897933E+032
  1.4142135623730950E+021
Double precision sum:    3.1415926536039354E+032
```

Matrix multiplication

Method	CPU usage	Performance
row-by-col	Contiguous	Efficient cache usage
col-by-row	Non contiguous	Poor cache locality
matmul	Contiguous	Optimized (BLAS subroutines)

- Fortran uses **column-major order** to store matrices, that’s why the second method is expected to be less efficient in terms of CPU than the other two

Column-major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

CPU performance:

- CPU time check
- Optimization flags (-O2, -O3): optimize matrix loops by reordering, unrolling and parallelizing them

Code development

Matrix-matrix multiplication loops:

```
! Matrix multiplication (row-by-column)
do i = 1, m
    do j = 1, p
        C(i,j) = 0.0_real32
        do k = 1, n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
        end do
    end do
end do
```

```
! Matrix multiplication (column-by-row)
do j = 1, p
    do i = 1, m
        C(i,j) = 0.0_real32
        do k = 1, n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
        end do
    end do
end do
```

CPU-time loop:

```
open(unit=10, file='row_col.dat', status="unknown", position="append")
call cpu_time(init_time)
call matmul_rowbycol(M1, M2, prod1, dim, dim, dim, .false.)
call cpu_time(end_time)
write(10,*) dim, end_time - init_time
close(10)

open(unit=10, file='col_row.dat', status="unknown", position="append")
call cpu_time(init_time)
call matmul_colbyrow(M1, M2, prod2, dim, dim, dim, .false.)
call cpu_time(end_time)
write(10,*) dim, end_time - init_time
close(10)

open(unit=10, file='matmul.dat', status="unknown", position="append")
call cpu_time(init_time)
prod3 = matmul(M1, M2)
call cpu_time(end_time)
write(10,*) dim, end_time - init_time
close(10)
```

Università degli Studi di Padova

Results