Introduction to High-Performance Computing

Exercise1

Giorgio Amati
Alessandro Ceci

Corso di dottorato in Ingegneria Aeronautica e Spaziale 2025
g.amati@cineca.it / g.amaticode@gmail.com
alessandro.ceci@uniroma1.it

Agenda

- ✓ Simple “warm-up” exercise: Matrix-Matrix Multiplication
- ✓ Complete the code
  - Fortran/C
- ✓ Check the results
- ✓ Extract some Performance figure (in MFLOPs)

You can use:

- ✓ Any available HW
- ✓ Any available compiler
- ✓ Any compiler option

Example: matrix-matrix multiplication

Simple problem: for 2 n^2 matrices we have to:

- ✓ compute n^3 products and n^3 sums
- ✓ load 2*n^2 data and to store n^2 data
  - Ratio computation vs. load/store is O(n)!

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

```
Fortran:          c(i,j) = c(i,j) + a(i,k)*b(k,j)
C:              c[i][j] = c[i][j] + a[i][k]*b[k][j];
```

- ✓ **MM multiplication is used for supercomputing rankings (top500)**

How to do

✓ Clone the repository

- `git clone https://github.com/gamati01/HPCLessons.git`

```
.
── LESSON1
│    ├── clean.sh
│    ├── compile.c.sh
│    ├── compile.fortran.sh
│    ├── EXERCISE1.pdf
│    ├── HPC-1.pdf
│    ├── inc_precision.h
│    ├── mm.c
│    ├── mm.F90
│    ├── mod_tools.F90
│    └── README
├── HPC-0.pdf
└── README.rst
```

✓ Complete the code

```
70 ! main loop
71     call system("date      > time.log")
72     call timing(time1)
73 !
74 ! write bottom here 3 nested loops....
75 !
76             c(i,j) = c(i,j) + a(i,k)*b(k,j)
77
78     call timing(time2)
79     call system("date      >> time.log")
```

✓ Complete the code

```
45    time1 = clock();
46    /*                                  */
47    /* write here 3 nested loop         */
48    /*                                  */
49                  c[i][j] = c[i][j] + a[i][k]*b[k][j];
50
51
52    time2 = clock();
```

compile.fortran.sh

- ✓ simple script to compile the code
- ✓ choose the available compiler & compiler options

```
rm -rf *.o mm.x *.mod
#
…
#
# gfortran (GNU)
COMP=gfortran
OPT=
#
echo "compiling with " $COMP $OPT
#
$COMP $OPT mod_tools.F90 -c
$COMP $OPT mm.F90 -c
$COMP $OPT mod_tools.o mm.o -o mm.x
#
echo "That's all folks!!!"
```

compile.c.sh

- ✓ simple script to compile the code
- ✓ choose the available compiler & compiler options

```
rm -rf *.o mm.x *.mod
#
…
#
# gcc (GNU)
COMP=gcc
OPT=
#
echo "compiling with " $COMP $OPT
#
$COMP $OPT mm.c -c
$COMP $OPT mm.o -o mm.x
#
echo "That's all folks!!!"
```

./mm.x (Fortran)

✓ Fortran: If coded correctly, it should give an output like that

```
 ---------------------------------------
  Matrix-Matrix Multiplication
  precision used              15
  rel. 0, naive multiplication
  Which matrix size?
1024
  Matrix size       =         1024
  Memory size (MB) =          24
 ---------------------------------------
 initialization   1.1718750000000000E-002
   0.4293334354359644         0.9410485065499756       0.00000000000000
 ----------------------------
 CPU:time for moltiplication  3.140625000000000
 CPU:MFLOPS                   683.7758861940298
 CPU:check                    257.1789318419338
```

✓ size (e.g. 1024) given by standard input

✓ check ~ size/4

./mm.x (C)

✓ C: If coded correctly, it should give an output like that

```
==============================
Enter the size
1024
 single precision
size 1024
==============================
Initialization
Elapsed time for initialization
Total time -----------------> 0.043431
==============================
Time -----------------> 7.044017
Mflops ----------------> 304.866335
Check -------------> 252.884160
```

✓ size (e.g. 1024) given by standard input

✓ check ~ size/4

Homework: Fill the table

Size	Fortran	C
256*256		
512*512		
1024*1024		
2048*2048		

- ✓ Compiler used:
- ✓ Compiler option used:
- ✓ HW used:

(OUR) HOMEWORK

My homework: `CoCalc`

Size	Fortran	C
256*256	8589	1877
512*512	4294	2589
1024*1024	4469	2594
2048*2048	2822	2631

- ✓ Compiler used: gnu (rel. 11.4)
- ✓ Compiler option used: **`-Ofast/-O3`** (gfortran/gcc)
- ✓ HW used: ?????

mm.F90 & mm.c

✓ Fortran

```
76  do k = 1, n
77       do j = 1, n
78            do i = 1, n
79                 c(i,j) = c(i,j) + a(i,k)*b(k,j)
80            enddo
81       enddo
82  enddo
```

✓ C

```
46 for (i = 0; i < nn; i++)
47       for (k = 0; k < nn; k++)
48            for (j = 0; j < nn; j++)
49                 c[i][j] = c[i][j] + a[i][k]*b[k][j];
```

my homework: Leonardo

Size	Size MB	Fortran MFlops	C Mflops
128^3	0.1875	357	493
256^3	0.75	477	511
512^2	3	473	494
1024^2	12	478	514
2048^2	48	470	511

- ✓ Compiler used: **`gfortran/gcc`** (rel. 12.2.0)
- ✓ Compiler option used: -
- ✓ HW used: Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz

my homework: `Leonardo/2`

Size	Size MB	Fortran MFlops	C Mflops
128^3	0.1875	-	-
256^3	0.75	8589	5660
512^2	3	7635	5606
1024^2	12	7635	5900
2048^2	48	5704	6301

- ✓ Compiler used: **`gfortran/gcc`** (rel. 12.2.0)
- ✓ Compiler option used: **`-Ofast`**
- ✓ HW used: Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz

my homework: `Leonardo/3`

Size	Size MB	Fortran MFlops	C Mflops
128^3	0.1875	-	-
256^3	0.75	8589	11225
512^2	3	13743	11266
1024^2	12	12494	10901
2048^2	48	12388	11043

- ✓ Compiler used: **`ifort/icc`** (rel. 23.2.0)
- ✓ Compiler option used: `-O3`
- ✓ HW used: Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz

my homework: Leonardo/4

Size	Size MB	Fortran MFlops	C Mflops
128^3	0.1875	-	-
256^3	0.75	8589	16836
512^2	3	7635	13944
1024^2	12	7967	11055
2048^2	48	7635	11388

- ✓ Compiler used: **`nvfortran/nvc`** (rel. 23.11.0)
- ✓ Compiler option used: `-O3`
- ✓ HW used: Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz

Take home message

- ✓ HW is important
- ✓ Coding is crucial
  - Data access has big impact on performance: why loop ordering affects so much performance?
- ✓ SW is important
  - Different compiler could present different performances
  - Compiler options can present very different performances: why?

- ✓ Where all these differences comes from?
- ✓ It is mandatory to understand how a computer really works!!!!!