Code Optimization
Hands-On

INAF HPC School 2025
Catania, Sep. 22nd - 26th

Luca Tornatore
O.A.T.S

Outline

Examples to play with:

- Memory access
- Conditional branches
- Exploiting everything in loops

How it works

Examples

- Every example code aims at clarifying a topic discussed in the lecture.
- We re-focus, and we get through the example together.
- Usually the example illustrates several ways in which you can code, from the naīve to a much better, so that you can understand it and verify.
- At the beginning, you can verify just by the shrinking of the run-time; we’ll teach you how to look at different metrics

How it works

Exercise

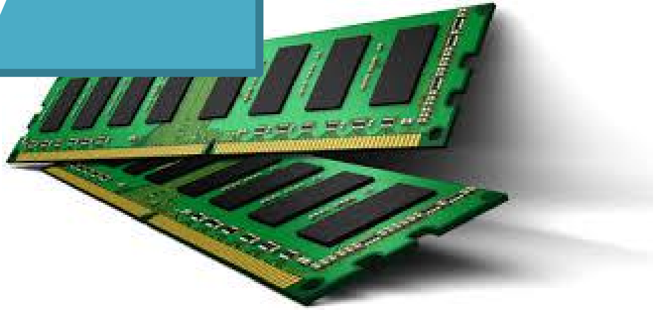
- We give you a problem to solve
- Tipically, we'll start from a (semi-complete) serial code and we'll ask you to parallelise it

How it works

All of us will be available to discuss, answer questions, solve technical problems and solve doubts.

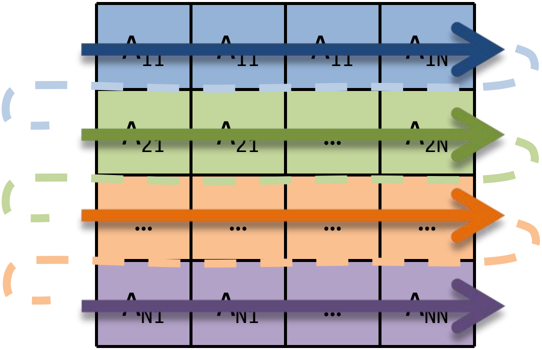
Please, rise your hand and call us at any moment.

Accessing the Memory

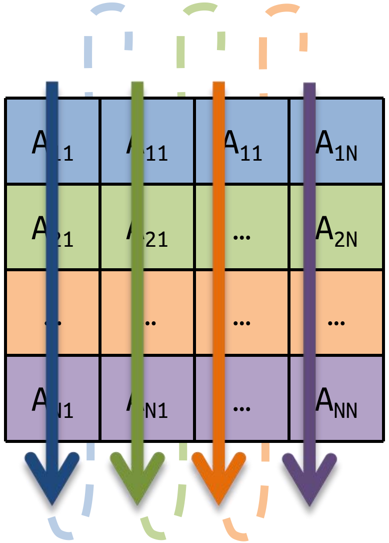


Array traversal

After having built a 2D array(*), let's assess what is the impact of accessing it by contiguous locations or by strided locations.
That is the very first step to sense how the cache works.



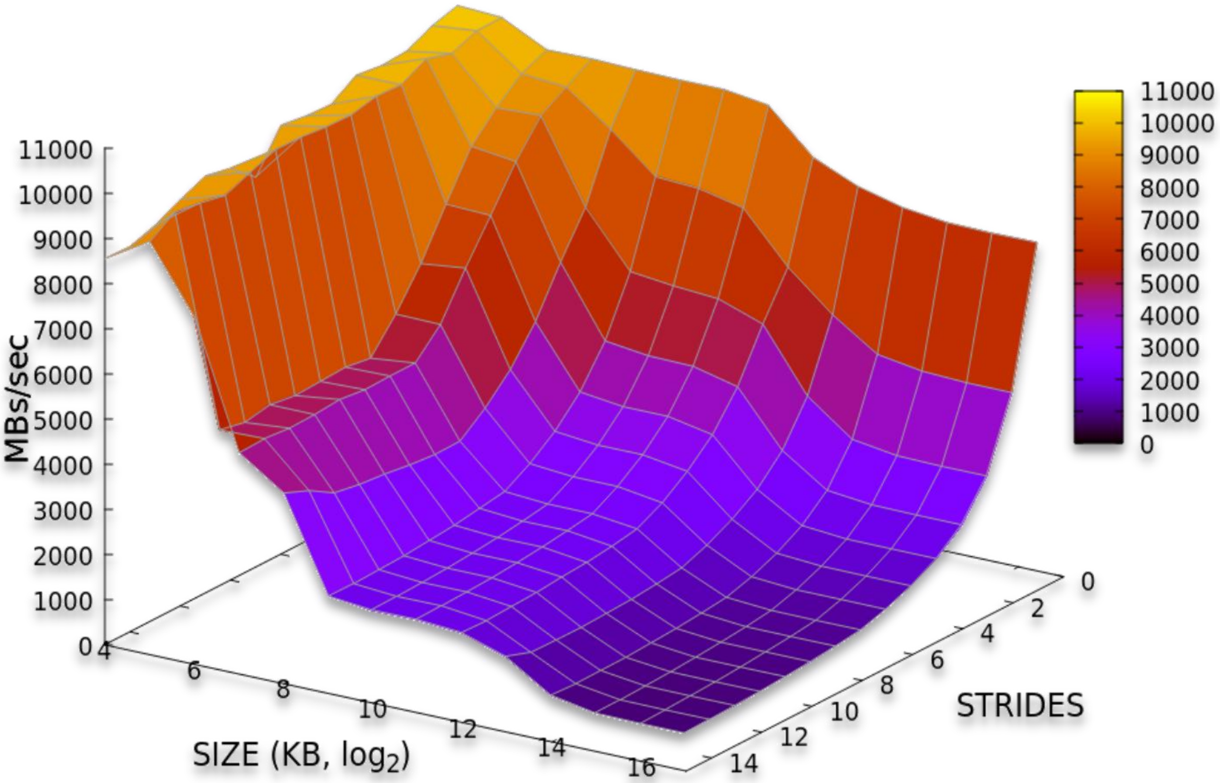
A_{11} A_{11} A_{11} A_{1N}
A_{21} A_{21} … A_{2N}
… … … …
A_{N1} A_{N1} … A_{NN}



A_{11} A_{11} A_{11} A_{1N}
A_{21} A_{21} … A_{2N}
… … … …
A_{N1} A_{N1} … A_{NN}

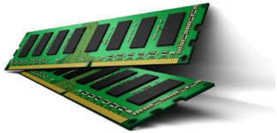
(*) how do you "build" a 2D array on an intrinsically 1D object like the RAM ?)

The memory mountain

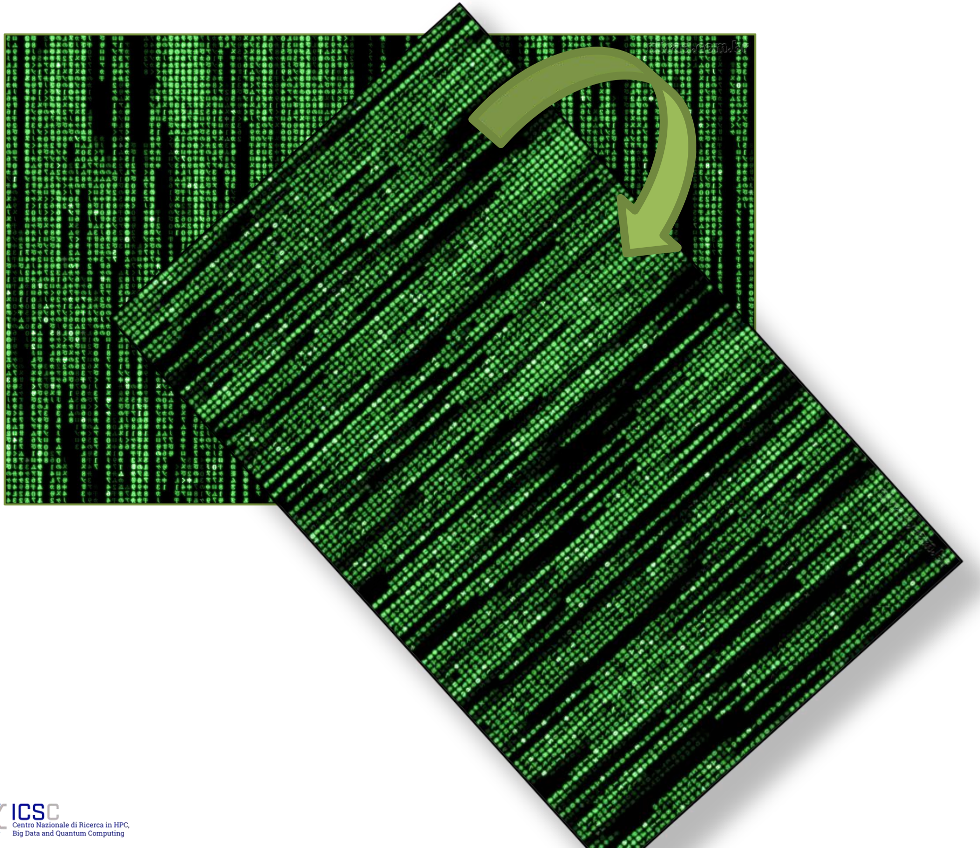


That is basically the extension of the previous exercise.

Let’s dig more, and get the mountain from your own laptop



Transpose a 2D Matrix



In this problem it is unavoidable to either write or read with a stride.

```
                 for ( int i = 0; i < Nrows; i++ )
                    for ( int j = 0; j < Ncols; j++ )
strided write →       At[j][i] = A[i][j]
strided read  →       At[i][j] = A[j][i]
```

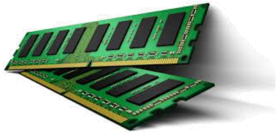
Which one is better ?
Is there any way to be more cache friendly?

Hot and Cold fields

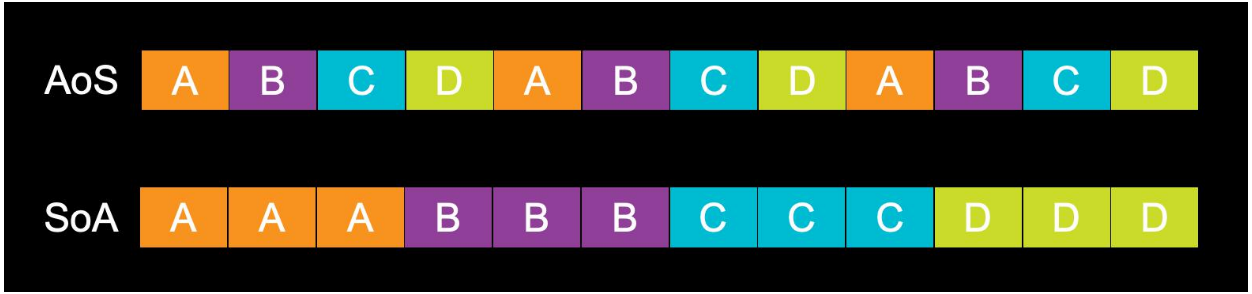
We use a much known data structure, the “linked list”, at its worst

(which is pretty easy, though.
Actually, we covertly aim at convince you *not to use* linked lists).

Why **conveying big data structures** around is a **bad idea** if you’re using only a tiny fraction of them *very often* - for instance, to retrieve the data to be processed - and most of them only once, or very few times?



AoS vs SoA



AoS	A	B	C	D	A	B	C	D	A	B	C	D
SoA	A	A	A	B	B	B	C	C	C	D	D	D

Which is the “best” data layout?
Well, it depends.

But let’s test AoS and SoA in a simple case, and in a the same case rendered slightly less simple.

Conditional Branches

Unpredictable stream

Branch predictors are surprisingly good… when there is something to predict.
If This Then That.

How do you do when the unpredictable jumps in?

| Unpredictable streams

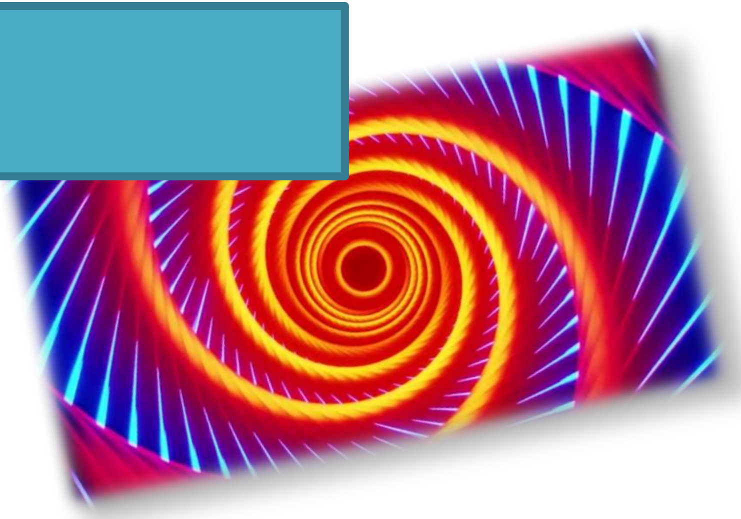
We will explore two cases:

1) scanning an array to process only a subset of its values

2) element-wise sorting two arrays so that

$$A_i \geq B_i \ \ \forall i$$

Loops



Matrix Multiplication

A great classic, we couldn’t miss it.
This problem, whose arithmetic intensity grows with the problem’s size, is used to rank supercomputers world-wide.

It is also among the bricks of AI.

Exploiting superscalarity

that’s all, have fun