Assignment II: Programming Languages for HPC (Ao Song)

Github: https://github.com/ao-song/hpc/tree/main/1

Part 2:

With row-major 2D array, every row of the matrix is placed continuously in computer memory, then the second and so on like the example in following picture.

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

In memory, each row will stay together like the following picture.

...	...	0,0	0,1	0,2	1,0	1,1	1,2	2,0	2,1	2,2	...

So the address offset to the first element will be:

*offset = index_of_row*number_of_column + index_of_column (1)*

likewise, the column-major layout will be similar but columns together. The layout of above example will be as following.

...	...	0,0	1,0	2,0	0,1	1,1	2,1	0,2	1,2	2,2	...

the address offset will be:

*offset = index_of_column*number_of_row + index_of_row (2)*

Performance wise, it is necessary to choose the iteration pattern of multi-dimension array based on the layout. For c which is row-major layout, it is important that we should iterate the matrix row by row. As each row is in continuous address in memory, the CPU should almost always be able to hit the cache during the iteration, which should significantly improve the performance.

c is row-major layout while fortran is column-major layout, we need be careful when making function calls between the languages, like in the coding assignment, when calling c function from fortran, if pass the address of first element, in the c code we need calculate the address in the column-major way to get a correct result. When locate the element in matrix, we should use the formula(2) shown above. Please find the details in the code.

Bonus (Julia):

After read this article and some other reference, my impression of Julia is that generally it is a dynamic language, like python, which is more flexible and easy to use than c and fortran but with very good performance in numerical computing area because of the optimized JIT technique which can work like a run-time template compiler.

A rich type system and multiple dispatch can make a Julia functions reusable and the program become much concise. One function can work with different types including self defined types. Julia's broadcasting is a very powerful concept which allows you to compactly and efficiently express an element-wise operation over containers and scalars by annotating with a dot. With Julia's fusion concept, which is instead of first constructing an intermediate object, then do the calculation, Julia can skip the intermediate object and do the operation in same time to be more efficient. For example, the expression `([1, 2, 3] .+ [10 20 30 40]) ./ 10`, in Julia, the addition and division for each element can be calculated the same time.

Julia is also column-major layout which make it possible to use some fortran's library like the LAPACK.

So basically Julia is more flexible and easy to use like python, but with higher performance. Julia is kind of self-contained for numerical computing. Python need some external help like pybind11 to call external implementations which in some case is not very convenient and not so easy to debug.

But Julia also has some disadvantages. The community of Julia is not as active as python. Sometimes it is not very easy to find a library to use. Julia is based on JIT, the booting time is normally slow comparing to c and fortran.