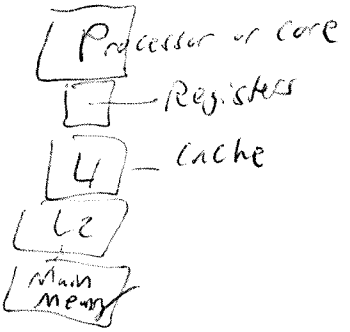
2/15/2022 (1)

Serial Computation

Performance is often not what is learned in theory classes
Performance is not an exact science
Performance is full of mysteries
Always we can learn something new

1 * Caches
2 * Row vs Column Major
3 * The Stack + the heap
4 * "Mutation" to avoid heap allocations
5 * Broadcasting (loop fusion)
6 * Views
7 * Type Inference
8 * Type Specialization
9 * Type Stability
10 * Multiple Dispatch
11 * Global Scope
12 * Static Arrays

1 Cache (pronounced "cash")



More expensive faster memory
Programs grab not only the item you ask for but also nearby items anticipating their use. A "cache miss" is a call to main memory.
The ability to re-use data in cache allows for faster programs.

MATMVL $O(n^3)$ ops / $O(n^2)$ data → $O(n)$ reuse
MATADD n^2 ops / n^2 data → no reuse

2 Row & column Major

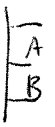
Julia storage

Python storage

3 Stack — Memory where compiler knows the size at compile time "statically allocated"

Heap - Compiler does not know Uses pointers "dynamically allocated"

Stack



Heap

allocations = creation of pointers

Many allocations can be a sign of performance loss

4 ~~Mutation~~

Changing an already existing data structure such as an array to avoid new allocations

5 Broadcasting "Loop Fusion"

* What is it?
* Read the "More dots" blog You'll learn many important truths

2/15/2022

Myth: Vectorization good - Loops Bad
(true in Python/MATLAB/R ...)
Truth: These languages are poor at devectorized code
Truth: These languages can be good at individual vectorized operations but lose out with fused loops

* Blog: Why vectorized code is not as fast as it could be
* Blog: Why Julia can broadcast arbitrary operations

In Julia it is okay to write loops
" " " " " to write vectorized

My opinion: Write what is cleaner
I find array operations are for grown ups often but not always

6. Views
Slices create allocations

7 Type Inference & the JIT

JIT = Just in Time Compiler
Wrong view = Compiler = magic thing that speeds up codes
Right View = Type inference + Type specialization
= Julia's core design feature

— continue with .jl file