

CMOR 420/520 Fall 2024

Weekly "short homeworks" or quizzes to ensure they're progressing along with the material

C exercises

- Write a swap code using pointers
- Write a code "getindex"
- Create a vector struct and "resize" and "concatenate" functions
 - The vector struct should have "int n" and "double * data" fields
 - The vector should be initialized to zero
 - Resize should use "realloc"
 - Concatenate should use "memcpy"

C assignment:

1. Create a MyMatrix struct which holds an (m-by-n) dense matrix
 1. Fields should be "int num_rows, num_cols" and "double ** data"
 2. For data, use Format 3 in class (contiguous memory and the pointer reassignment trick)
 3. Create a function "initial_MyMatrix(int m, int n)"
2. Implement a "matmul" routine between two MyMatrix instances.
 1. It should return an "int" error code, where "0" is no error, and "1" indicates size compatibility error.
 2. The output should be passed in as a MyMatrix argument
 3. Test for correctness: compute $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$
 1. Answer should be
 1. 70 80 90
 2. 158 184 210
 3. 246 288 330
 2. Compute and print out the maximum absolute difference between the exact and computed matrix
 3. Put this in "test_matmul.c"
3. Compare runtimes (use the minimum time over 25 samples) timing
 1. Compute runtimes for two square matrices with sizes from 10 to 512.
 2. Put the timing code in "time_matmul.c". This code should output the timings. Run this for both gcc -O0 and gcc -O3
 3. Plot the runtimes for both gcc -O0 and gcc -O3 using your preferred plotting software (Matlab, Python, Julia, Excel, etc) and include them in your writeup.
4. Code organization and compilation
 1. Make sure your code is readable: at minimum, indent your code properly

- and provide comments when necessary.
2. All struct definitions and function prototypes should be placed in "include/MyMatrix.h"
 3. All function definitions should be placed in "src/MyMatrix.c"
 4. Modify the provided Makefile from class to compile both "test_matmul.c" and "time_matmul.c"
 5. Written questions:
 1. Explain whether it would or would not be possible to implement a contiguous-in-memory column major format in C using the same pointer reassignment trick as Format 3.
 2. Suppose you wished to write functions to append one MyMatrix to an existing MyMatrix either vertically or horizontally. Which one would be easier and why? Be specific.
 6. Grad students:
 1. Implement a "reshape(int new_m, int new_n, MyMatrix * A)" routine. This should:
 1. Modify the input MyMatrix *
 2. Keep the underlying data layout
 3. Check compatibility of dimensions
 4. Test correctness of your implementation in "test_reshape.c". Provide build instructions for this in your writeup.

C++ assignment

- Implement a template linear algebra library
 - Create a MyVector type with overloaded vector space operations
 - Create a MyMatrix type with overloaded vector space operations.
 - Implement both a const and non-const indexing operator overload
 - Implement matrix-matrix and matrix-vector multiplication using your MyMatrix and MyVector classes
 - Define A, B, b, x . Compute $b += A * B * x + (A - B) * b$
- Grads: compare your implementation of "matmul" against eigen.h or BLAS.
 - Compare your templated MyMatrix and MyVector classes runtime with -O3 to your C code in Assignment 1
 - Compare against Eigen

Julia assignment

- Choose one matrix types out of:
 - A rank "r" rectangular matrix
 - Block diagonal matrix where blocks are identical up to scaling
 - A matrix type you propose (instructor permission required)
- Writeup and coding:
 - Explain how to efficiently represent these matrices, and then implement

- them in Julia's abstract array interface.
 - You do not need to implement `setindex!`.
 - Additionally, explain how to take advantage of the structure of your matrix to implement matrix multiplication. Then, actually implement the `mul!(C, A, B)` function in Julia.
- Testing:
 - Demonstrate that you can run `x += 2 * A * x + sum(sin.(A), dims=2)`.
 - Demonstrate that you can beat the speed of Julia's built-in matrix multiplication for certain matrix parameters. Report the matrix size and matrix parameters that you used.

Weekly schedule:

- ✓ Week 15: Programming in Julia: wrapping up Julia, metaprogramming, Assignment 4 "due"
 - ✓ Opt: JuMP.jl
 - ✓ ODEs: OrdinaryDiffEq.jl
- ✓ Week 14: Assignment 4 announced (Thanksgiving)
- ✓ Week 13: Programming in Julia: important Base Julia packages (LinearAlgebra, SparseArrays, Plots), creating a Julia package, important packages in the Julia ecosystem
 - ✓ Tabular data: DelimitedFiles.jl, DataFrames.jl
 - ✓ Arrays: StaticArrays.jl, StructArrays.jl, FillArrays.jl
 - ✓ AD: ForwardDiff.jl
- ✓ Week 12: Programming in Julia: interfaces, the AbstractArray interface, type instabilities, allocations, efficient programming in Julia, the package manager, Project.toml files, (Assignment 3 due)
- ✓ Week 11: Starting Julia: overview, installation, the REPL and shell environments, macros, how Julia works, broadcasting, multiple dispatch, custom types, abstract types, parametric types
- ✓ Week 10: Programming in C++: templates, template programming and specialization, the standard library (vectors, pairs, map, iterators), linear algebra libraries, error handling (exceptions).
- ✓ Week 9: Programming in C++: rule of three, dynamic memory in C++, inheritance, inheritance issues (virtual functions), operator overloading, forward declarations, const vs non-const functions
- ✓ Week 8: Programming in C++: differences from C, references, function overloading, classes, custom constructors and destructors (Assignment 2 due)
- ✓ Week 7: Programming in C and C++: const keyword, linking external libraries,

- valgrind and gdb, I/O, Start C++. (I will miss Friday lecture)
- ✓ Week 6: Programming in C: macros, compilation / linking, Makefiles
- ✓ Week 5: Programming in C: pointers and dynamic memory management, structs
- ✓ Week 4: Programming in C: control flow, function definitions, header files, pointers, arrays
- ✓ Week 3: Github, LaTeX (mention Rice's premium Overleaf access), concepts in numerical computing
- ✓ Week 2: Linux, Github
- ✓ Week 1: intro, Linux, Linux exercises