Classes, debugging and optimisation

Paolo Joseph Baioni

March 26, 2025

Exercise 1 - What you need to know

- The implemented matrix class is organized as **column-major**, *i.e.* $A(i,j) =$ `data[i + j * rows()]`, conversion from 1d to 2d indexing is performed by the utility method `sub2ind`.
- Access to elements is implemented both in `const` and non-`const` versions, by overloading `operator()`.
- Data is private, *getter methods* expose what is needed to the user, both `const` and non-`const` versions are provided.
- Naive implementation of matrix-matrix multiplication is slow because it has low *data locality*, simply transposing the left matrix factor improves performance significantly[1].
- The `#include <ctime>` header provides timing utilities, `tic()` and `toc(x)` macros start and stop the timer.

[1] See *M. Kowarschik, C. Weiß. (2002). Lecture Notes in Computer Science. 213-232. DOI: 10.1007/3-540-36574-5_10* for further details.

Exercise 1.1

Starting from the provided implementation of the class for dense matrices (and column vectors represented as 1-column matrices) based on `std::vector`, implement the following methods:

- transpose: $A = A^T$.
- `operator*`: matrix-matrix and matrix-vector multiplication.

Exercise 1.2

- Transpose the first factor in matrix multiplication before performing the product.
- Compare the execution speed with respect to the previous implementation.

Exercise 1.2 - Details

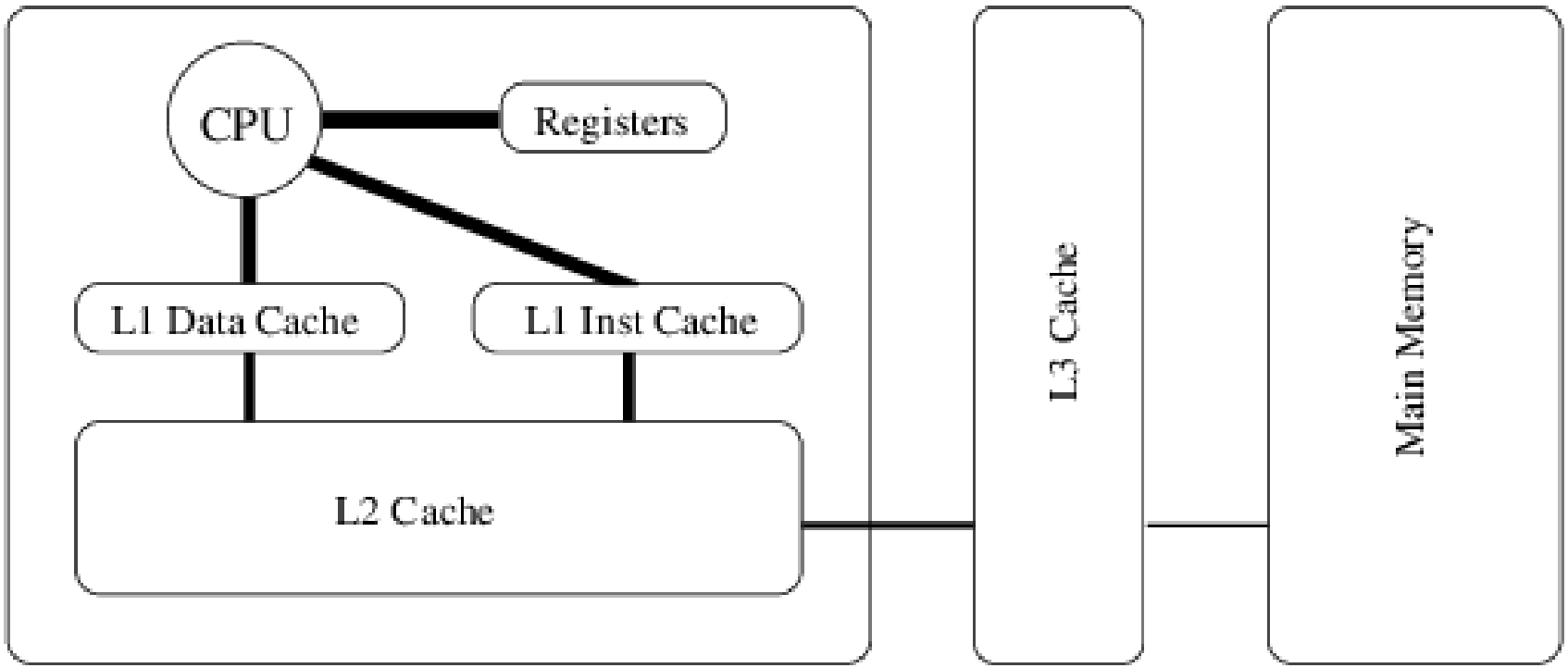

Figure: Typical memory layout of a computer.

Exercise 1.2 - Details

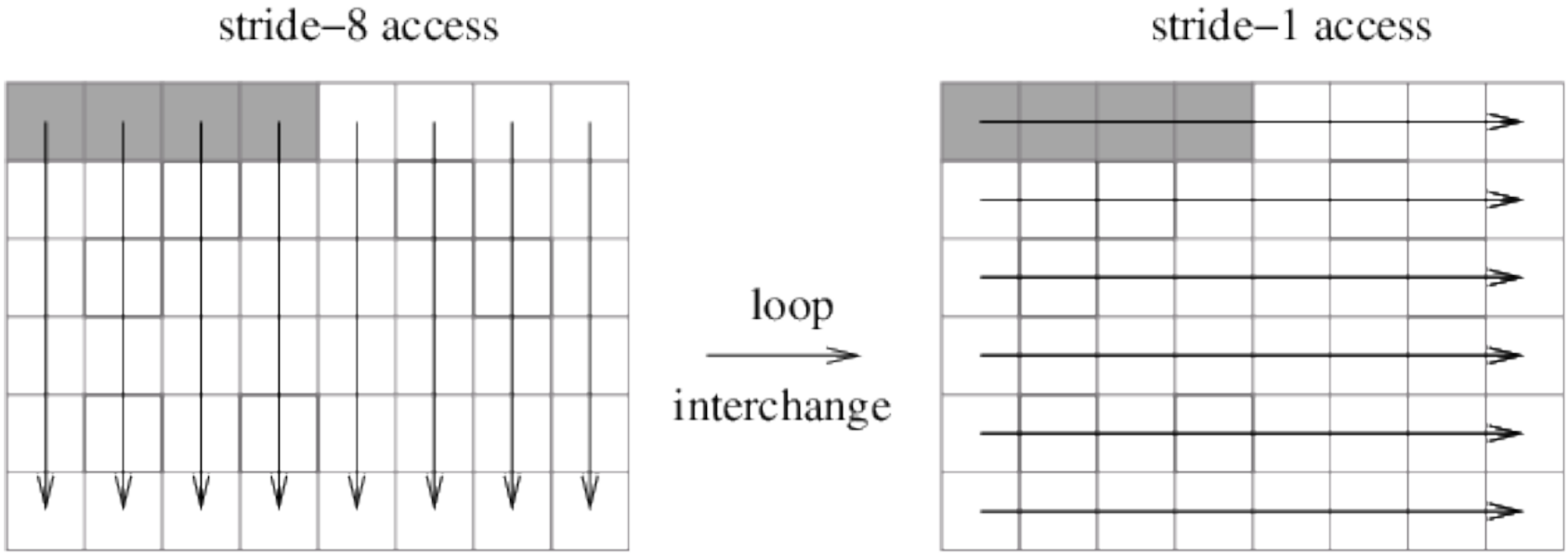

Figure: Example with a row-major matrix.

Exercise 1.2 - Details

```
1: double sum;
2: double a[n, n];
3: // Original loop nest:
4: for j = 1 to n do
5:    for i = 1 to n do
6:       sum+ = a[i, j];
7:    end for
8: end for
```

```
1: double sum;
2: double a[n, n];
3: // Interchanged loop nest:
4: for i = 1 to n do
5:    for j = 1 to n do
6:       sum+ = a[i, j];
7:    end for
8: end for
```

Figure: Example with a row-major matrix.

Exercise 1.3

- Include the `Eigen/Dense` header.
- Use the `Eigen::Map` template class to wrap the matrix data and interpret it as `Eigen::MatrixXd`.
- Compare the execution speed with respect to the previous implementations.

Bonus

- Link to Openblas
- Use the `DGEMM` function to perform the matrix-matrix product
- `matrix-0.4/matrix.{h,cpp}`

See pacs-Labs/Labs/2019/03-cachealignment/matrix-0.4 for the solution

Exercise 1.4

Going back to Ex. 1.2, perform the subsequent analysis:

- coverage (`lcov`)
- memcheck (`valgrind`)
- profile (`valgrind, kcachegrind`)

Exercise 2

The program `integer-list` in the directory `02-bug` has:

- a compile error;
- a run-time error;
- a memory leak;
- bonus: a potential memory leak that is not captured by the `main`.

Find all the issues and fix them.

Get help by using `gdb` and `valgrind`.

The directory `02-bug-solution` contains the fixed code, please don't look at it before trying to solve the exercise by yourself!