High-Performance Computing - Challenge

Master in Statistics for Data Science

Last generated: 2024-02-19

uc3m | Universidad Carlos III de Madrid

Problem statement

As the Wikipedia article explains,

> **Matrix chain multiplication** (or the **matrix chain ordering problem**) is an optimization problem concerning the most efficient way to multiply a given sequence of matrices. The problem is not actually to *perform* the multiplications, but merely to decide the sequence of the matrix multiplications involved. The problem may be solved using dynamic programming.

The issue comes from the fact that matrix multiplication is associative, as it is regular scalar multiplication. This means that, if A, B, C are matrices, there are two possible ways[1] of computing the product. The result will be the same, but the **computational complexity** may be quite different. The product of a matrix with dimensions $X \times Y$ and a matrix with dimensions $Y \times Z$ requires $X \cdot Y \cdot Z$ scalar product operations. If, for example, A is a 60×5 matrix, B is 5×30, and C is 30×10, then computing

[1] $(AB)C = A(BC)$

- $(AB)C$ requires $60 \cdot 5 \cdot 30 + 60 \cdot 30 \cdot 10 = 27000$ operations;
- $A(BC)$ requires $5 \cdot 30 \cdot 10 + 60 \cdot 5 \cdot 10 = 4500$ operations.

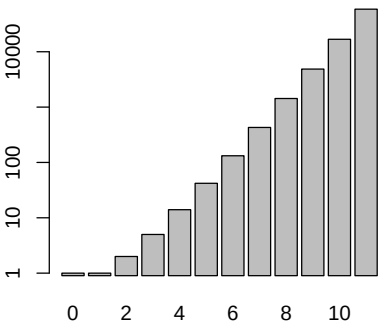
That is, the second case achieves around a **83% reduction** in complexity compared to the naive approach of just multiplying the matrices in order. For three matrices this is more or less straightforward. But the number of possible parenthesizations grows quickly with the number of matrices. For four matrices, there are five options:

$$((AB)C)D = (A(BC))D = (AB)(CD) = A((BC)D) = A(B(CD))$$

In general, the number of ways to parenthesize an expression of $n+1$ terms is given by the n-th Catalan number[2] C_n, i.e., for $n = 0, 1, 2, 3, ...$, the sequence (A000108 in the OEIS):

$$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, ...$$

[2] $C_n = \frac{1}{n+1}\binom{2n}{n}$



In summary, the problem is ***how to determine the optimal parenthesization of a product of n matrices***? There are several solutions in the literature. Brute-forcing the problem (checking each possible option) would require a run-time that is exponential in the number of matrices, which is slow and impractical for large n. A dynamic programming approach achieves $O(n^3)$ complexity, and there are other approaches that achieve $O(n \log n)$. There are also approximate solutions that run in $O(n)$.

Assignment

The assignment can be divided in several tasks:

- **Task 1**: Play with random matrices of different dimensions and the matrix product in R to find a compelling example of the multiplication of 5 matrices in which the naive default ordering[3] takes considerably more time than some custom parenthesization. Show this with a benchmark. Analyze the number of operations for each case and discuss the results.
- **Task 2**: Implement a C++ function that takes 5 matrices as input and multiplies them using RcppArmadillo without any parenthesization. Compare it with the previous ones in R (no parenthesization, best parenthesization). Discuss the results.
- **Task 3**: Implement a C++ function that takes a list of matrices and a vector defining the multiplication order, and performs the operations in the specified order using RcppArmadillo. Compare it with the previous case for the best parenthesization.
- **Task 4**: Investigate further in the scientific literature about the *matrix chain ordering* problem, specifically about the available algorithms. Implement **at least one algorithm** that solves the problem and produces a vector defining the multiplication order. Use it in conjunction with the function from task 3 to compare the approach of solving the matrix chain ordering problem *and* perform the product in the resulting order vs. R's best and worst cases (task 1) vs. Armadillo's naive approach (task 2).

[3] `A %*% B %*% C %*% D %*% E`

Treat the last task as a **very open** one, in the sense that you may produce several iterations with several optimizations.[4] You may start by solving the problem in R, then switching to C++; you may implement the dynamic programming algorithm in $O(n^3)$, then compare it with a solution in $O(n \log n)$; you may start with a sequential implementation, then try to improve it with parallelization methods from chapter 3... Task 4 is **your opportunity to *shine***.

[4] Beating this or that implementation is **not** the purpose, but to think critically about the results and **learn** along the way.

Delivery

Given the open-ended nature of the assignment (especially the last task), and since we are particularly interested in the discussion of the results, the deliverable will be:

- A `.Rmd` document with all the analyses, discussion and inline code.[5] Please also be sure to add a section of references with a list of bibliographical resources consulted.
- The knit'ed document (HTML or PDF, depending on the selected format).

[5] Note that R Markdown documents support the inclusion of C++ code via ```` ```{Rcpp} ```` chunks.

The documents will be delivered via the corresponding assignment in Aula Global **before Monday March 18**, 2024.