Task 1 - Language Benchmark of matrix multiplication
Big Data

Alberto Guedes

October 2025

Contents

Abstract

The following report presents the development and execution of benchmarking tests for matrix multiplication implemented in three different programming languages: C, Java, and Python. The main objective is to compare their performance in terms of execution time and computational efficiency when performing large-scale numerical operations. The results obtained provide insights into the trade-offs between language design, memory management, and runtime performance, offering a deeper understanding of how each language handles intensive computational workloads. The source code can be found in the following repository: GitHub Repository

1 Methodology

1.1 Implementations

The three implementations compute $C \leftarrow A \times B$ using the classical triple loop.

- C: static arrays allocated as 2D blocks. Time measured with a high-resolution monotonic clock.
- Java: arrays of `double[]` with warm-up on the first run. Built with Java 21.
- Python: lists-of-lists using pure Python loops. Time measured with `time.perf_counter()`.

1.2 Benchmark harness

For each language and matrix size $n \in \{10, 50, 100, 256, 512\}$, 10 repetitions have been run, printing a CSV row per run. Wall time is the primary metric. CPU and memory are also emitted:

- Python/Java: CPU as percent (%), memory as MB (RSS-like).
- C: CPU reported as accumulated CPU seconds, memory in MB.

CSV artifacts are stored under `results/`. A companion notebook including data visualization (`Benchmark_results.ipynb`).

2 Results

Table 1 shows mean wall time (seconds) for each language and size (10 runs per size). C is the fastest at all sizes; Java trails C by a small constant factor; Python exhibits orders of magnitude slower runtime as n grows, consistent with interpreter overhead.

Table 1: Mean wall time (seconds) by language and matrix size.

oprule Language	$n = 10$	$n = 50$	$n = 100$	$n = 256$	$n = 512$
C	0.000002	0.000108	0.001332	0.021497	0.180225
Java	0.000804	0.000923	0.002767	0.025719	0.285948
Python	0.000102	0.011126	0.088605	1.542429	14.234368

CPU and memory metrics follow in Tables 2 and 3. Note that C reports CPU in seconds, whereas Python/Java report percent; the values are therefore not directly comparable across languages. Within-language trends still inform relative scaling.

Table 2: Mean CPU metric by language and size. Python/Java in percent (%), C in CPU seconds.

oprule Language	$n = 10$	$n = 50$	$n = 100$	$n = 256$	$n = 512$
C (s)	0.00017	0.01074	0.13233	2.14209	17.82012
Java (%)	1.00	11.50	27.78	15.14	12.35
Python (%)	5.00	12.94	14.97	18.71	17.23

Table 3: Mean memory usage (MB) by language and size.

oprule Language	$n = 10$	$n = 50$	$n = 100$	$n = 256$	$n = 512$
C	0.00	0.00	0.00	0.00	0.00
Java	0.09	0.02	0.05	0.09	0.05
Python	0.00	0.01	0.03	0.24	4.74

Figure 1 (produced by `Benchmark_results.ipynb`) visualizes mean time, CPU, and memory versus size on log-scaled axes for readability.

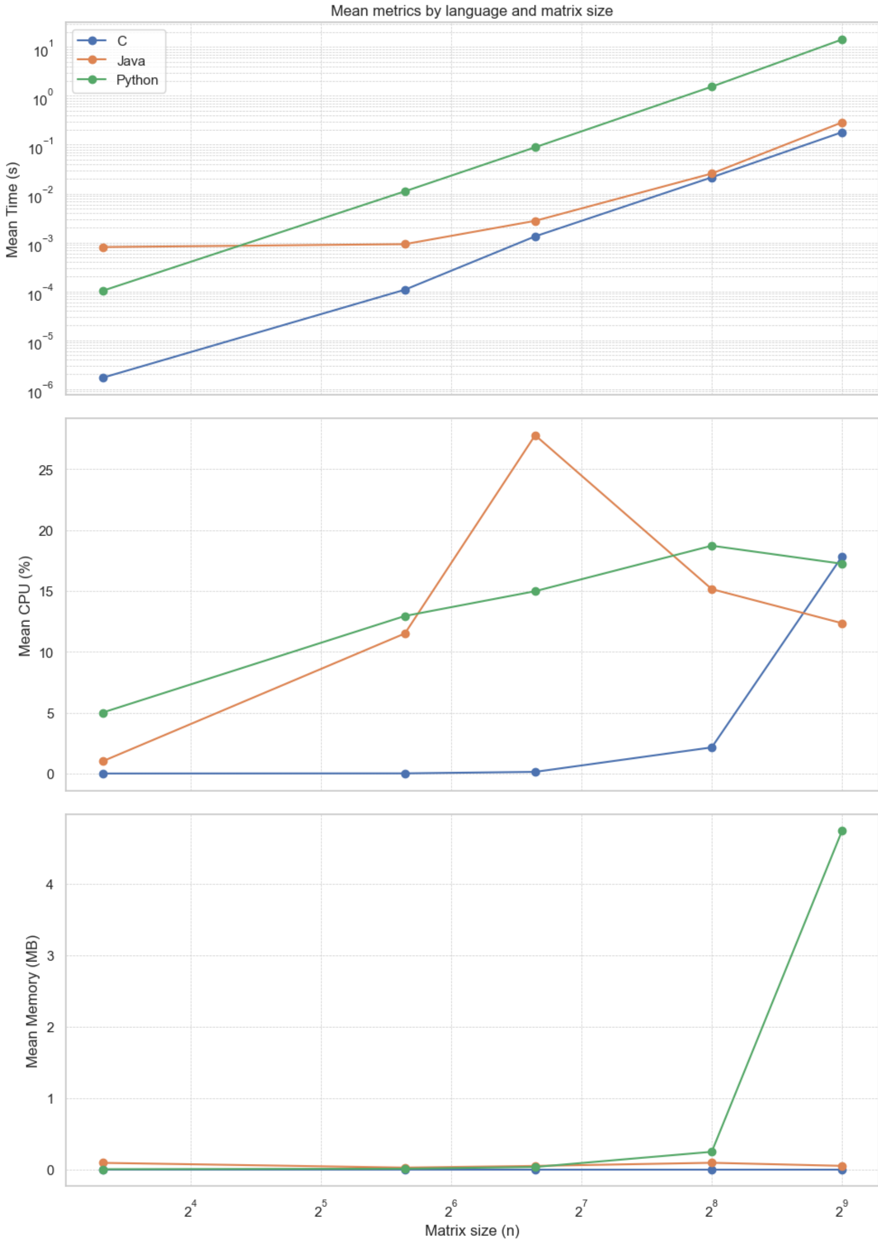

Figure 1: Benchmark results for C, Java and Python implementations.

3 Discussion

3.1 Runtime

The naive C implementation consistently outperforms Java and Python due to ahead-of-time compilation and minimal overhead. Java is competitive, particularly at small to mid sizes, but remains slower than C. Python's interpreter overhead dominates at larger n, yielding multi-order-of-magnitude slowdowns.

3.2 CPU and memory

Java and Python show modest CPU percentages per run, reflecting short single-threaded bursts per measurement. C's CPU metric (seconds) increases roughly with $\mathcal{O}(n^3)$ work, tracking wall time. Python memory grows with n as expected from list-of-list allocations; Java's reported per-run memory increments remain small given JVM reuse and GC behavior during runs; C's per-run deltas are effectively negligible at this scale.

3.3 Threats to validity

Results depend on compiler flags, JVM warm-up, Python interpreter version, and system load. CPU metrics are not normalized across languages (seconds vs percent) and should not be compared directly. Cache effects and memory layout (row-major access patterns) may bias certain loop orders. The implementations are intentionally naive and not vectorized nor parallelized.

4 Conclusion

C delivers the best performance for naive matrix multiplication, Java follows at a small constant factor, and Python is suitable only for small sizes unless vectorized libraries (NumPy) are used. For production workloads, prefer optimized libraries or low-level languages; for pedagogy or prototyping, Python remains convenient but slow.

Reproducibility

Benchmarks were run with 10 repetitions per size using the scripts and programs in this repository. Raw CSVs are under `results/`. The notebook `Benchmark_results.ipynb` computes aggregates and produces comparative plots.