Comparison of Matrix Multiplication

Vojtěch Kuzdas
Mo 14.30
Software Engineering
kuzdavoj@fel.cvut.cz

I. Methodology

A. Warm-up

The warmup period was determined by plotting the runtime of all three matrix multiplication implementations. Implementations were run in 3 forks of 30 iterations each. Matrices themselves were filled with integer numbers in the interval of [0, 1000]. Dimensions of matrices were selected to be 600x400 and 400x600.
Period was selected from the plot subjectively by the author. Author is of the opinion that first two iterations always deviated the most. The plots of all three implementations are displayed in ch. III. A.

B. Measurements

Java was used to implement the multiplications and OpenJDK's Microbenchmark Harness was used to measure the multiplications. Author performed 10 forks and 30 iterations of each kind of matrix multiplication (warmup excluded), each iteration measuring the average time of the given implementation. Matrices' dimensions were selected as before.

C. Comparison

Measured means of each implementations were compared to each other according to the recommended source (Section 10.1). This was computed using a python script. Resulting comparisons can be seen in the *results.json* file.

II. Machine specification

Measurements were done on author's personal windows laptop Zephyrus G14 with AMD Ryzen 7 CPU with 2.9-4.2 GHz and 16GB RAM.
No JVM parameters were used.

III. Results

A. Warm-up

See ch. I. A. for warm-up determination explanation.

Implementation	Forks	Iterations	Determined warm-up iterations
Multiply	3	30	2
MultiplyTrans	3	30	2
Multiply1D	3	30	2

Table 1: warm-up iterations determination

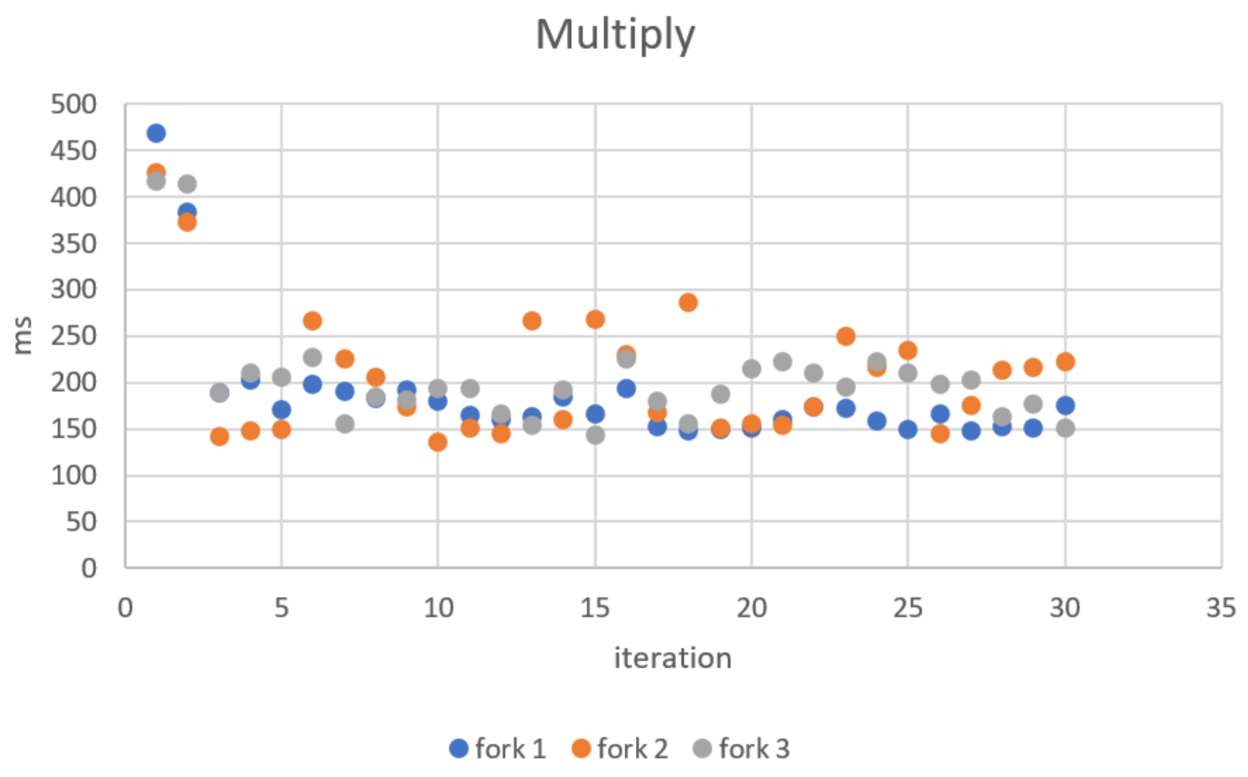

Figure 1: runtime of classic matrix multiplication

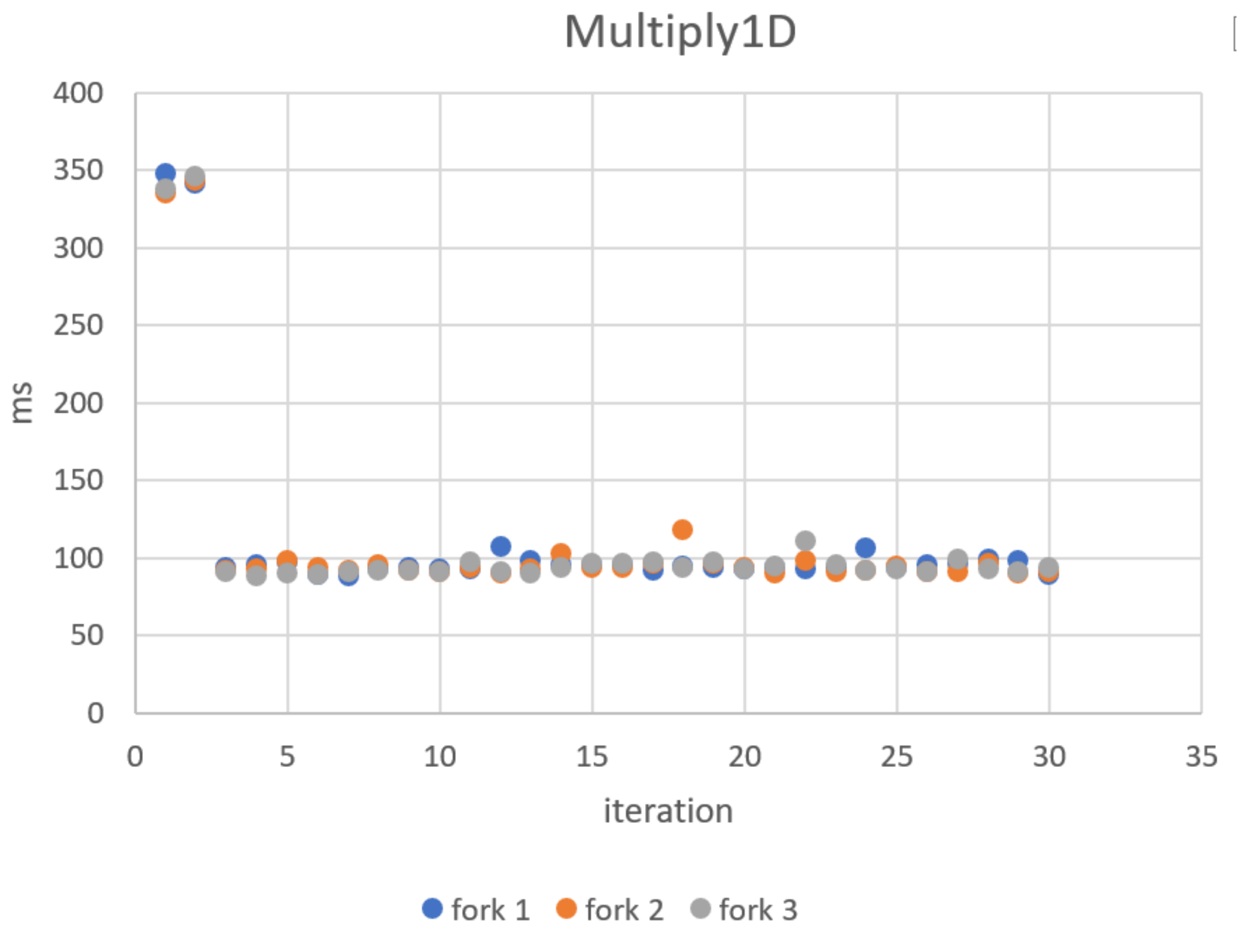

Figure 2: runtime of transposed matrix multiplication

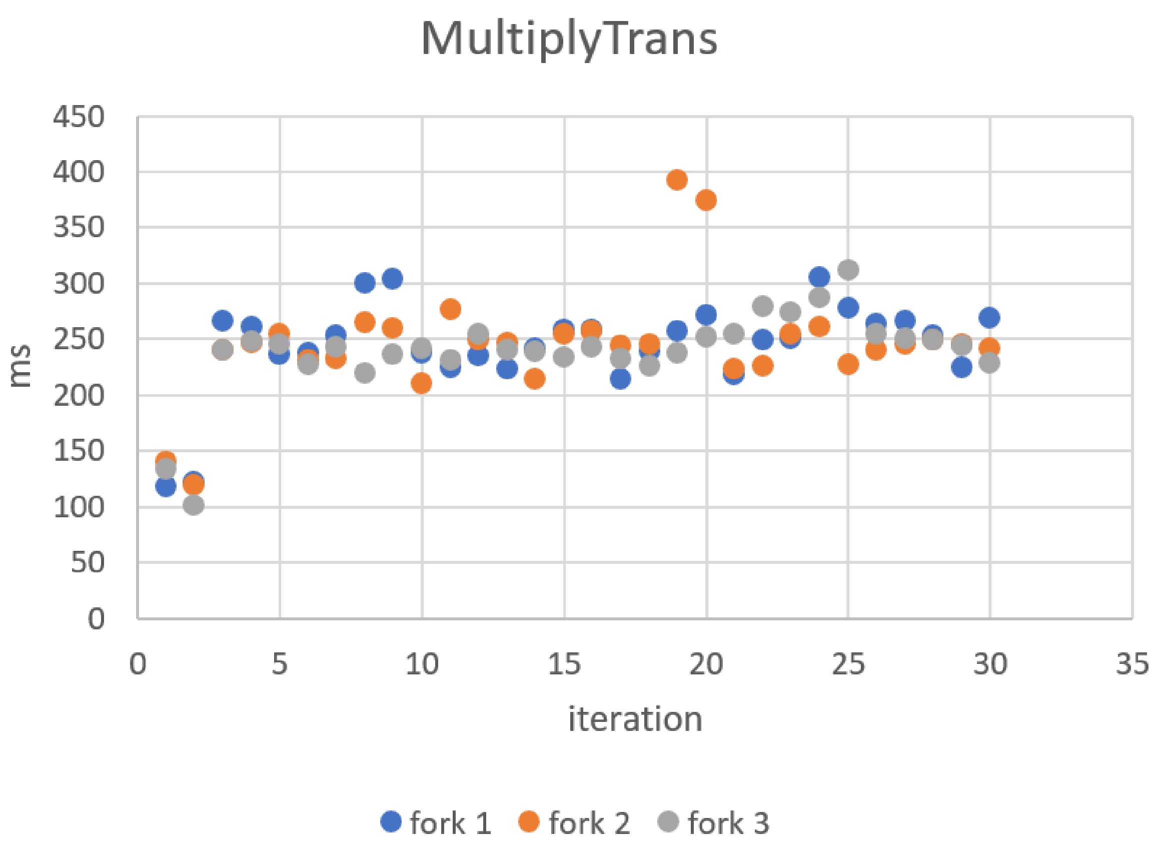

Figure 3:runtime of 1-D matrix multiplication

B. *Measurements*

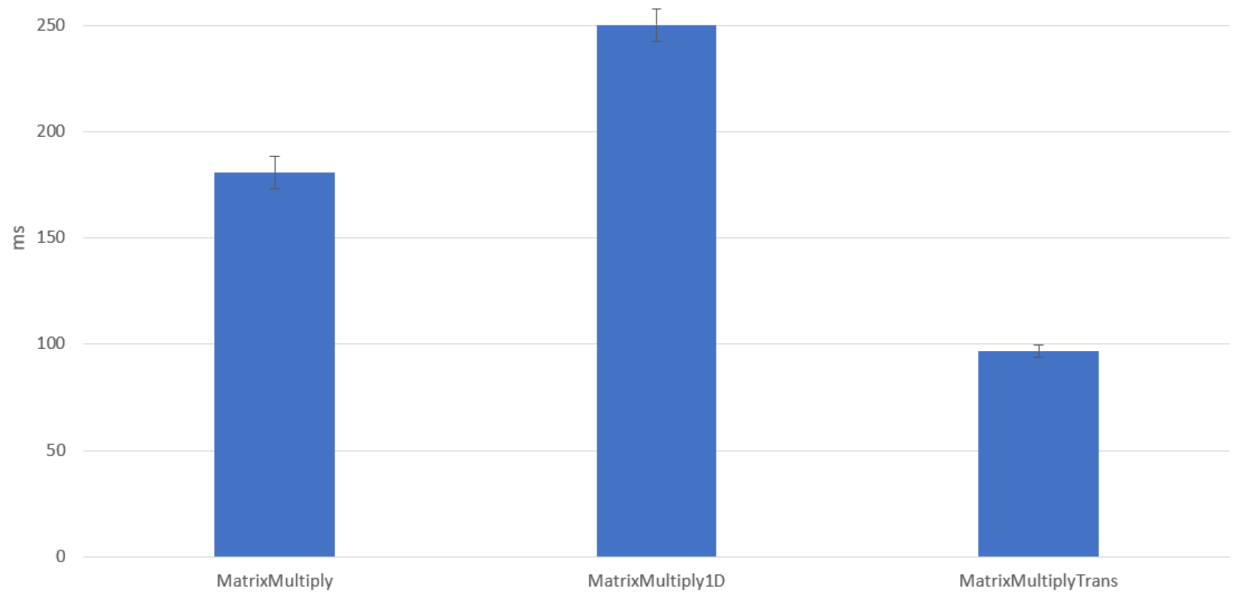

Figure 4: mean runtime of each implementation with confidence intervals

implementation	**mean**	**error**
MatrixMultiply	180.7547	7.596936
MatrixMultiply1D	250.1642	7.64641
MatrixMultiplyTrans	96.71852	2.739101

Figure 5: mean runtime of each implementation with confidence intervals

C. *Comparison*

Comparison	**Ratio**	**Lower bound**	**Upper bound**
Multiply x 1D	1.386447585	1.314418481	1.458476689
Multiply x Trans	0.536028538	0.508870029	0.563187047
1D x Multiply	0.723211075	0.68556191	0.760860241
1D x Trans	0.386976996	0.370911602	0.40304239
Trans x Multiply	1.870373677	1.77560902	1.965138334
Trans x 1D	2.588594391	2.480753861	2.69643492

Figure 6: comparison table

IV. Conclusion

Author concludes that the transposed implementation of matrix multiplication resulted in best performance given the mentioned machine specification. Furthermore, transposed multiplication was twice as fast as matrix multiplication in one-dimensional array – which was surprising for the author. Finally, transposed multiplication was almost twice as fast as classic matrix multiplication.