Matrix Multiplication Comparison of Different Approaches

High Performance Computing and Modern Architectures, 2022

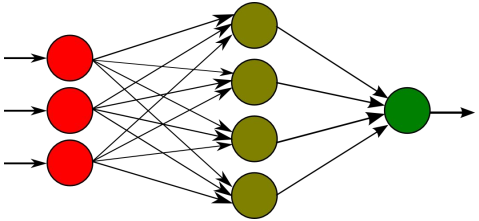
Done by: Nikita Kuznetsov, MSc-1 ACS

CPU Configuration

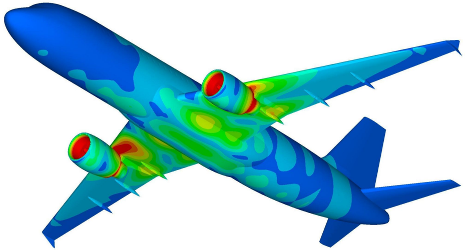
```
nikita@nikita-NBLK-WAX9X:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   43 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       AuthenticAMD
CPU family:                      23
Model:                           24
Model name:                      AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx
Stepping:                        1
Frequency boost:                 enabled
CPU MHz:                         1400.000
CPU max MHz:                     2100,0000
CPU min MHz:                     1400,0000
BogoMIPS:                        4192.22
Virtualization:                  AMD-V
L1d cache:                       128 KiB
L1i cache:                       256 KiB
L2 cache:                        2 MiB
L3 cache:                        4 MiB
NUMA node0 CPU(s):               0-7
```

Applications

- Deep Learning / Machine Learning



- Scientific modelling problems



- Economical and statistical problems

And more, more, more...

Matrix Multiplication

$O(n^2)$ Theoretical limit for general case (Strassen Hypothesis)

$O(n^{2.3727})$ Current record

Straightforward approach:

$$O(n^3)$$

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

Strassen Algorithm

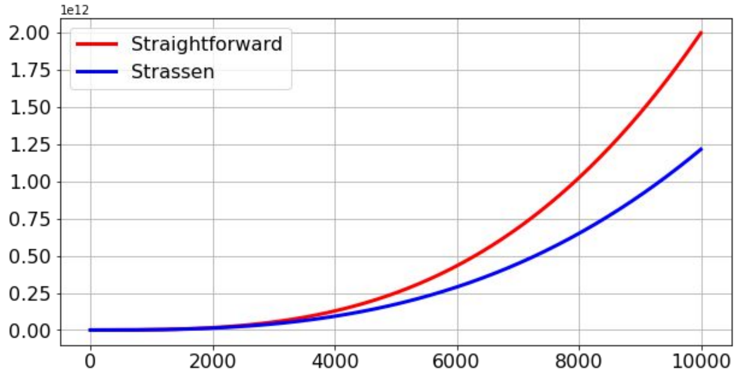
Complexity:

$$O(n^{log_2 7}) \approx O(n^{2.81})$$

$$
\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22});\\
M_2 &= (A_{21} + A_{22})B_{11};\\
M_3 &= A_{11}(B_{12} - B_{22});\\
M_4 &= A_{22}(B_{21} - B_{11});\\
M_5 &= (A_{11} + A_{12})B_{22};\\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12});\\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}),
\end{aligned}
$$

7 multiplications, 18 additions
vs
8 multiplications, 4 additions

$$2n^3 > 7n^{\log_2 7},$$
$$n > 667,$$



$$
\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}
$$

Parallel Strassen Algorithm

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

*Parallel matrix multiplication

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}); \\
M_2 &= (A_{21} + A_{22})B_{11}; \\
M_3 &= A_{11}(B_{12} - B_{22}); \\
M_4 &= A_{22}(B_{21} - B_{11}); \\
M_5 &= (A_{11} + A_{12})B_{22}; \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}); \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}),
\end{aligned}$$

Root 0

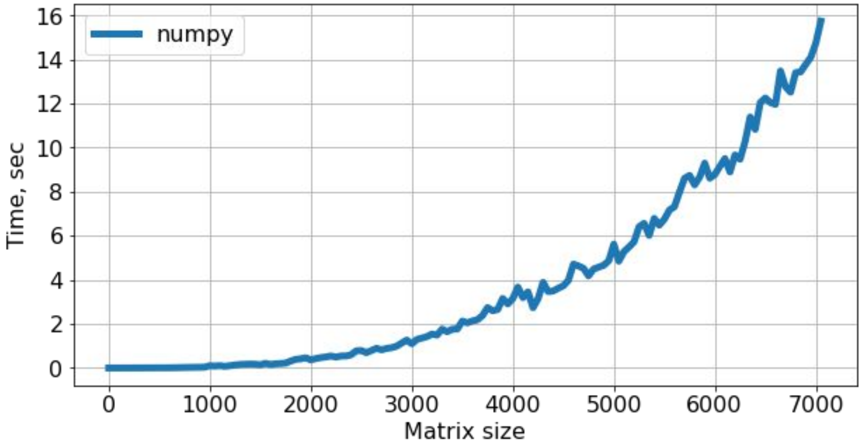
Gather

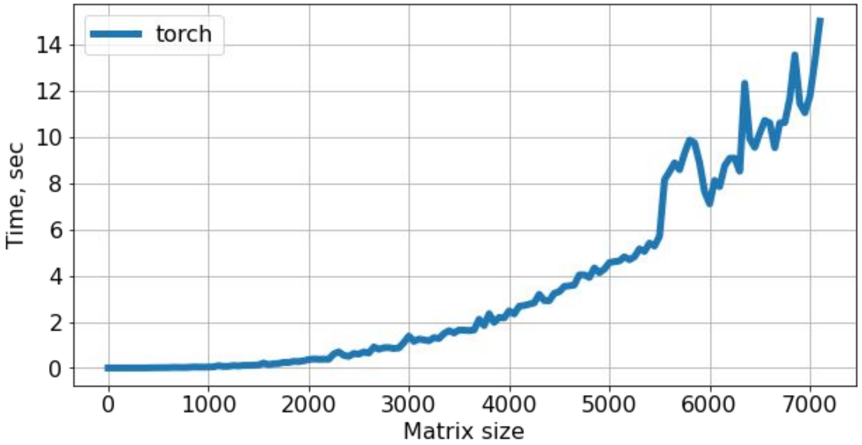
$$\begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}$$

Gather

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Python Libraries

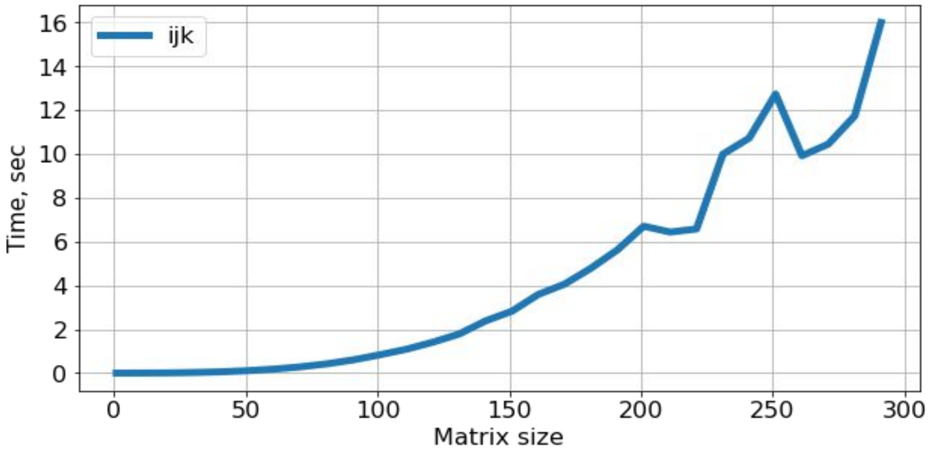


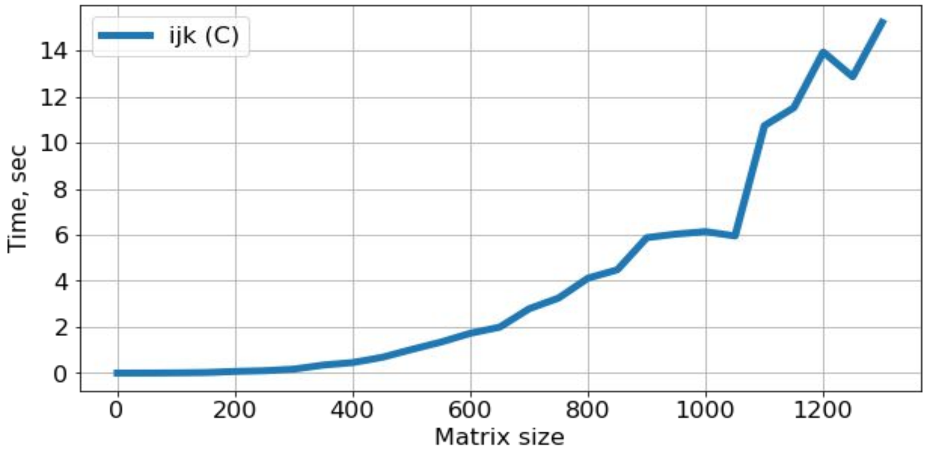


Max N = 7051 (numpy) ≈ **Max N = 7101 (pytorch)**

ijk-multiplication

```
for i in range(N):
        for j in range(N):
            for k in range(N):
                result[i, j] += A[i, k] * B[k, j]
```

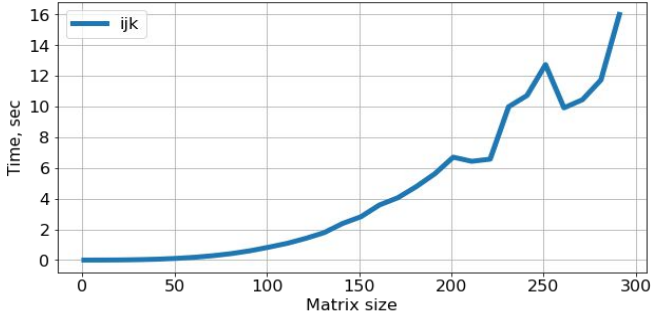


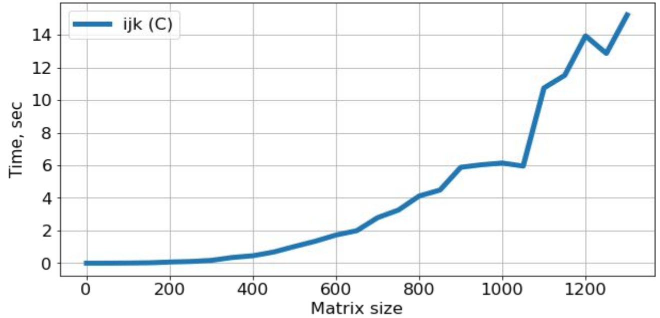


Max N = 291 (python) < **Max N = 1301 (C)**

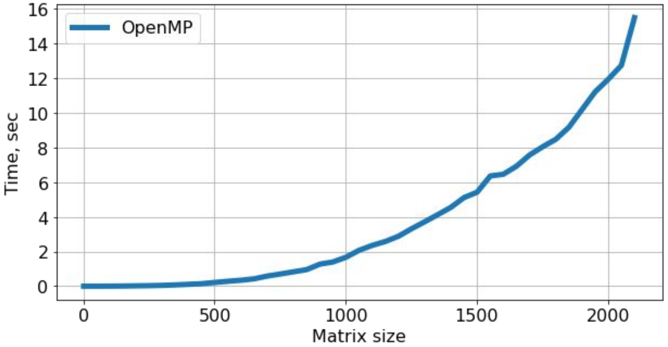
Almost 4.5 times

ijk-multiplication





Max N = 291 (python)



Max N = 1301 (C)

Almost 7.3 times

Max N = 2101 (OpenMP)

Almost 1.5 times

Strassen Algorithm

Matrix Size	Python (s)	MPI, N = 4 (s)	MPI, N = 8 (s)
32	0.132	0.002127	0.002042
64	0.865	0.003075	0.002336
128	6.012	0.007286	0.004207
256	29.174	0.028571	0.020210
512	211.437	0.162296	0.142489
1024	1373.739	2.352059	1.565214

Conclusions

- Python boosted by Numpy perform the most efficient
- Naive approach in Python is much slower comparing to C
- OpenMP helps to improve naive algorithm performance
- Second place in performance is taken by C Strassen algorithm
- Despite this fact, Strassen algorithm is not easy parallelizable and have to be carefully treated (from my observations)
- If something perform nicely, just use it :)

Matrix Multiplication Comparison of Different Approaches

High Performance Computing and Modern Architectures, 2022

Done by: Nikita Kuznetsov, MSc-1 ACS