Final Project
INF236: Parallel Programming
Parallel Matrix Multiplication

Kateřina Čížková, Luca Klingenberg

May 11, 2021

1 Introduction

Our goal was to implement parallel matrix multiplication. The standard algorithm has complexity $\mathcal{O}(n^3)$ and is straightforward to parallelize. We used its implementation as a reference. We compared it with Strassen's algorithm, which has complexity $\mathcal{O}(n^{\log 7})$ but is harder to implement.

2 Files

- `driver.c` driver allocates matrices, runs the multiplication, measures time and verifies results
- `driver.h` includes C libraries
- `cFiles.h` includes `.c` files with multiplication implementations
- `sequential_matmul.c` sequential simple matrix multiplication
- `sequential_strassen.c` sequential Strassen's algorithm
- `parallel_matmul.c` simple matrix multiplication in parallel
- `parallel_strassen_2_layers.c` two levels of Strassen's algorithm, then simple multiplication in parallel
- `parallel_strassen.c` recursive Strassen's algorithm with parallel additions and subtractions
- `z_order.c` functions for reordering matrix to z-ordering and back

3 Algorithms

In this section all the implemented algorithms are explained in further detail.

3.1 Matrix multiplication

The standard $\mathcal{O}(n^3)$ algorithm is simple. Each element c_{ij} of the resulting matrix $\mathbf{C}$ is calculated as $c_{ij} = \sum_k a_{ik} \cdot b_{kj}$. It is possible to implement the calculation with only three nested for-loops. But matrices are stored row-wise, so it is better to access it in consecutive order, in order to avoid cache misses. Because of that, we used the following implementation. Complexity is still $\mathcal{O}(n^3)$, but in practice it runs faster than with three nested loops that are indexed in kij order.

Algorithm 1 matrix multiplication

Input: $\mathbf{A}, \mathbf{B}$
Output: $\mathbf{C}$ (the resulting matrix)

function MATMUL($\mathbf{A}, \mathbf{B}$)
  for $i = 0, \ldots, n-1$ **do**
    for $j = 0, \ldots, n-1$ **do**
      $c[i][j] = 0$
    for $k = 0, \ldots, n-1$ **do**
      for $j = 0, \ldots, n-1$ **do**
        $c[i][j] += a[i][k] \cdot b[k][j]$
  return C

3.2 Parallel matrix multiplication

We just parallelized the outermost loop of the simple matrix multiplication algorithm described in previous section with `#pragma omp parallel for`.

3.3 Strassen's algorithm

The matrix multiplication can be formulated in terms of block matrices:

$$\mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} \\ \mathbf{A}_{10} & \mathbf{A}_{11} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} \\ \mathbf{B}_{10} & \mathbf{B}_{11} \end{pmatrix} = \begin{pmatrix} \mathbf{A}_{00}\mathbf{B}_{00} + \mathbf{A}_{01}\mathbf{B}_{10} & \mathbf{A}_{00}\mathbf{B}_{01} + \mathbf{A}_{01}\mathbf{B}_{11} \\ \mathbf{A}_{10}\mathbf{B}_{00} + \mathbf{A}_{11}\mathbf{B}_{10} & \mathbf{A}_{10}\mathbf{B}_{01} + \mathbf{A}_{11}\mathbf{B}_{11} \end{pmatrix} = \begin{pmatrix} \mathbf{C}_{00} & \mathbf{C}_{01} \\ \mathbf{C}_{10} & \mathbf{C}_{11} \end{pmatrix} = \mathbf{C}$$

According to this formula, it needs 8 multiplications of half-size matrices. The idea of Strassen's algorithm is to use more additions and subtractions, but only 7 multiplications half-size matrices. Then, the algorithm is used recursively for these multiplications. You can see a pseudocode of Strassen's algorithm below.

Algorithm 2 Strassen's matrix multiplication

Input: $\mathbf{A}, \mathbf{B}$
Output: $\mathbf{C}$ (the resulting matrix)

function STRASSEN($\mathbf{A}, \mathbf{B}, n$)
  if n == cutoff **then**
    return MATMUL($\mathbf{A}, \mathbf{B}$)
  $\mathbf{P}_1 = \text{STRASSEN}(\mathbf{A}_{00} + \mathbf{A}_{11}, \mathbf{B}_{00} + \mathbf{B}_{11}, \frac{n}{2})$
  $\mathbf{P}_2 = \text{STRASSEN}(\mathbf{A}_{10} + \mathbf{A}_{11}, \mathbf{B}_{00}, \frac{n}{2})$
  $\mathbf{P}_3 = \text{STRASSEN}(\mathbf{A}_{00}, \mathbf{B}_{01} - \mathbf{B}_{11}, \frac{n}{2})$
  $\mathbf{P}_4 = \text{STRASSEN}(\mathbf{A}_{11}, \mathbf{B}_{10} - \mathbf{B}_{00}, \frac{n}{2})$
  $\mathbf{P}_5 = \text{STRASSEN}(\mathbf{A}_{00} + \mathbf{A}_{01}, \mathbf{B}_{11}, \frac{n}{2})$
  $\mathbf{P}_6 = \text{STRASSEN}(\mathbf{A}_{10} - \mathbf{A}_{00}, \mathbf{B}_{00} + \mathbf{B}_{01}, \frac{n}{2})$
  $\mathbf{P}_7 = \text{STRASSEN}(\mathbf{A}_{01} - \mathbf{A}_{11}, \mathbf{B}_{10} + \mathbf{B}_{11}, \frac{n}{2})$
  $\mathbf{C}_{00} = \mathbf{P}_1 + \mathbf{P}_4 - \mathbf{P}_5 + \mathbf{P}_7$
  $\mathbf{C}_{01} = \mathbf{P}_3 + \mathbf{P}_5$
  $\mathbf{C}_{10} = \mathbf{P}_2 + \mathbf{P}_4$
  $\mathbf{C}_{11} = \mathbf{P}_1 - \mathbf{P}_2 + \mathbf{P}_3 + \mathbf{P}_6$
  return C

There are 18 additions and subtractions calculated in Strassen's algorithm. We used Winograd's version of Strassen's algorithm described in [**boyer2009memory**]. It needs only 15 additions and subtractions because some results are reused. We also used scheduling table 1 from the same paper. Therefore our implementation needs only two temporary matrices in each recursive call to store intermediate results.

Strassen's algorithm is recursive and it is working with quarters of input matrices. Because of that, it is useful to use some recursive ordering for storing matrices in memory, for example z-ordering (figures 1, 2).

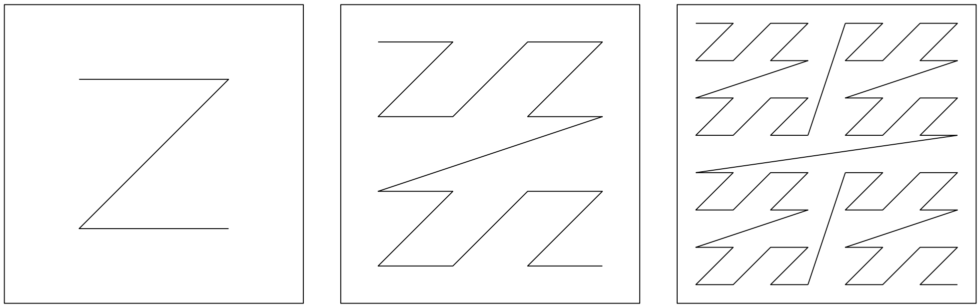


Figure 1: z-ordering diagram

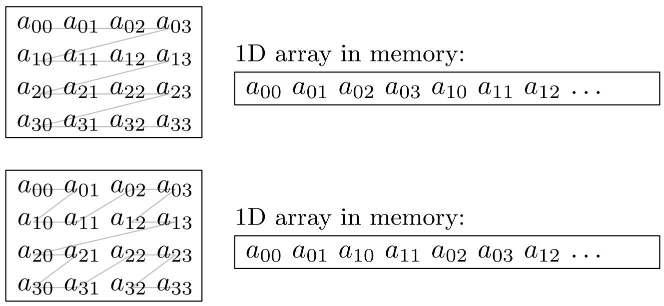


Figure 2: row-ordered and z-ordered matrix stored in memory

Its advantage is, that all submatrices are stored in consecutive parts of memory. However, there is switch from Strassen's algorithm to the simple matrix multiplication when the matrices are too small so Strassen's algorithm overhead is too big. For the simple matrix multiplication is better to have the matrices ordered row-wise. In our implementation, we combined both z-ordering and usual row-by-row ordering. We used the z-ordering for bigger submatrices, but we left the small submatrices which are multiplied with the simple multiplication algorithm in row-wise order (figure 3). Functions for reordering matrices to this layout and back are implemented in `z_order.c` file.

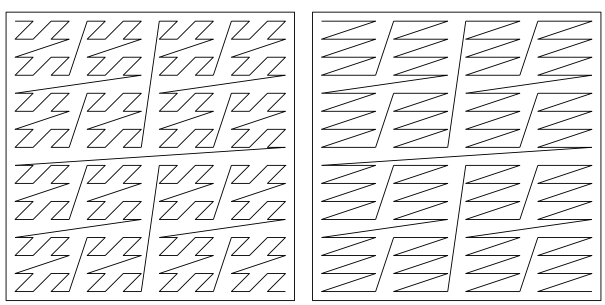


Figure 3: z-ordering only for bigger submatrices

3.4 Parallel Strassen's algorithm

It is straightforward to parallelize additions and subtractions in Strassen's algorithm. The problem is that for small matrices it is not worth it to initialize a parallel region. Because of that, we can not just insert `#pragma omp parallel for` before the addition and subtraction for-loop.

Then, we were thinking about parallelizing recursive calls – calculate $\mathbf{P}_1$ to $\mathbf{P}_7$ in parallel. But since we are using only two temporary matrices in each recursive call, it is necessary to finish a recursive call before starting the next one. This may be solved using more space for intermediate results, but we did not try that.

We decided to try some number of recursive steps and than switch to the parallelized simple multiplication.

3.4.1 2-layers parallel Strassen's algorithm

This implementation uses two levels of Strassen's algorithm (hardcoded as functions `parallel_strassen_level_1` and `parallel_strassen_level_2`). There are seven parallel regions initialized for additions and subtractions in level 1 and only one for all additions, subtractions and multiplications in level 2.

3.4.2 Recursive parallel Strassen's algorithm with matrix size cutoff

After some measurements with 2-layers parallel Strassen's algorithm, we implemented this version. The idea is same, but there are more recursive calls of Strassen's algorithm with parallelized additions and subtractions (similar to `parallel_strassen_level_1`). When is reached cutoff matrix size, it is switched to the parallelized simple multiplication. We tried some cutoffs and we think that the best one is 512×512.

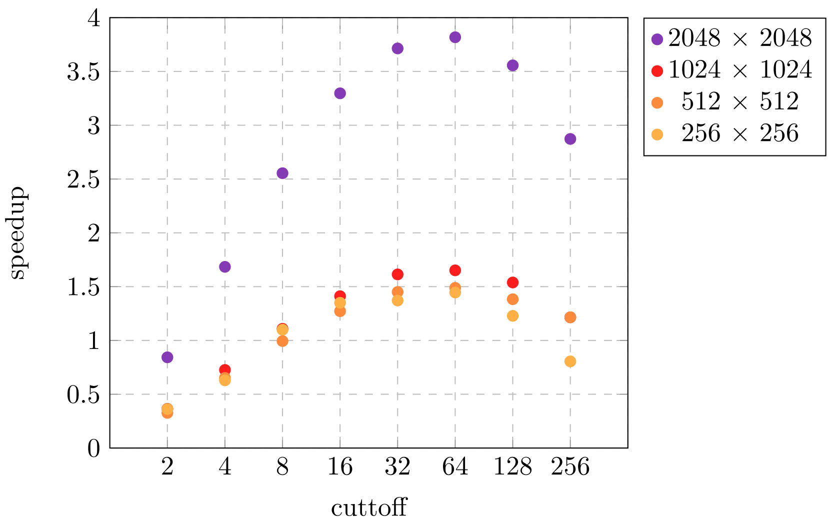
Since there are at least two layers of Strassen's algorithm and we set the cutoff to 512×512, we need to solve cases when the matrices are smaller than 2048×2048. We use only the the parallelized simple multiplication in these cases.

4 Experiments

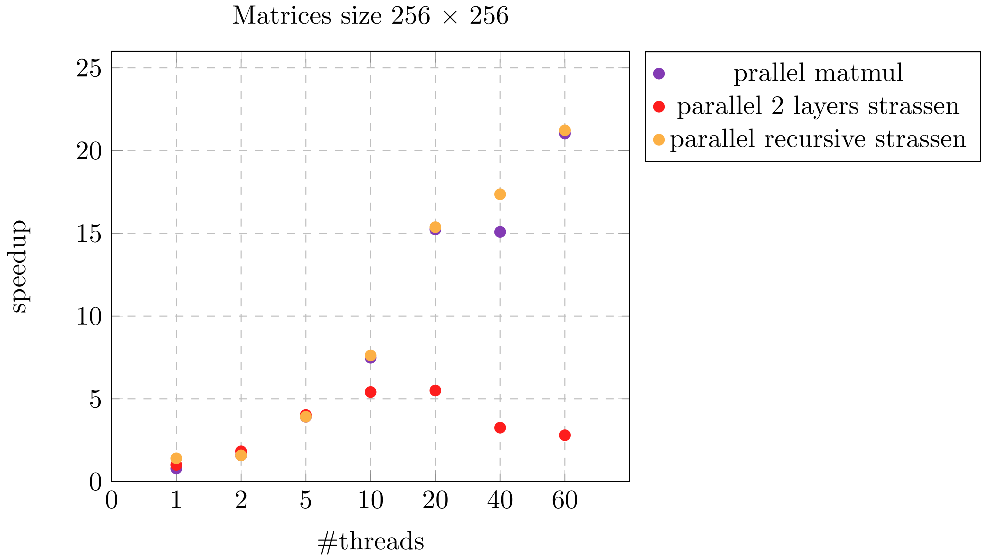
We run all tests on brake.ii.uib.no machine.

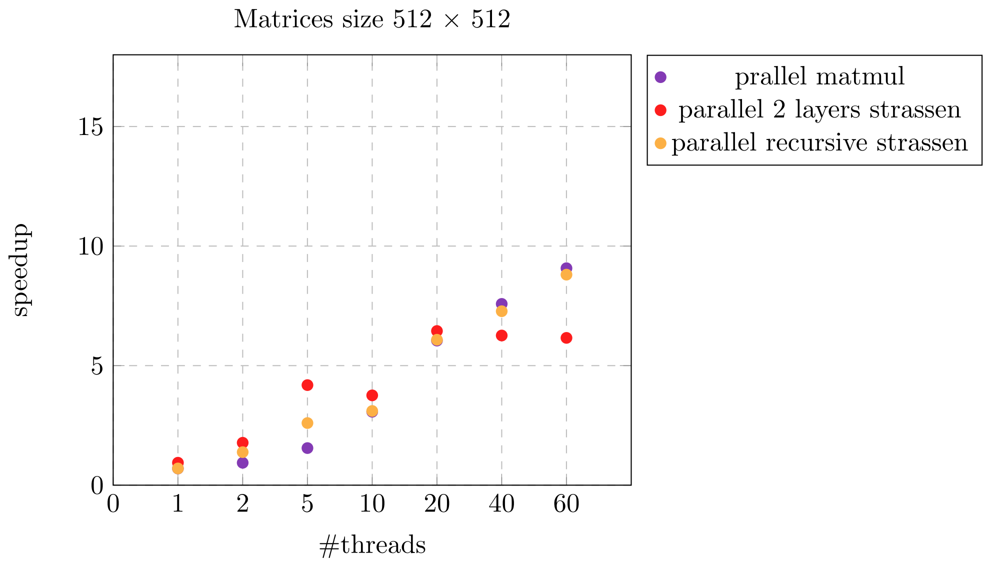
4.1 Sequential Strassen's algorithm

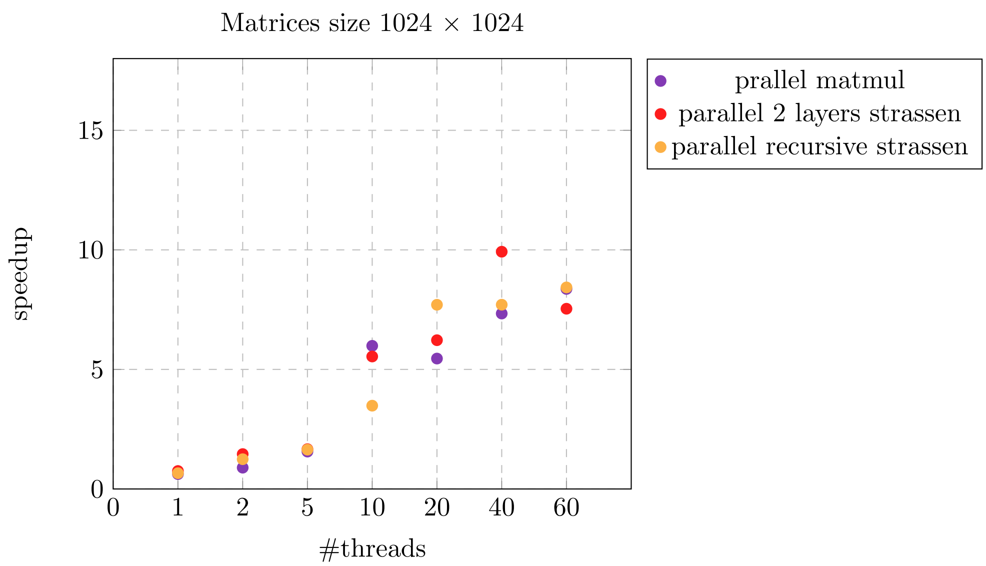
Sequential Strassen's algorithm with different values for the cutoff level



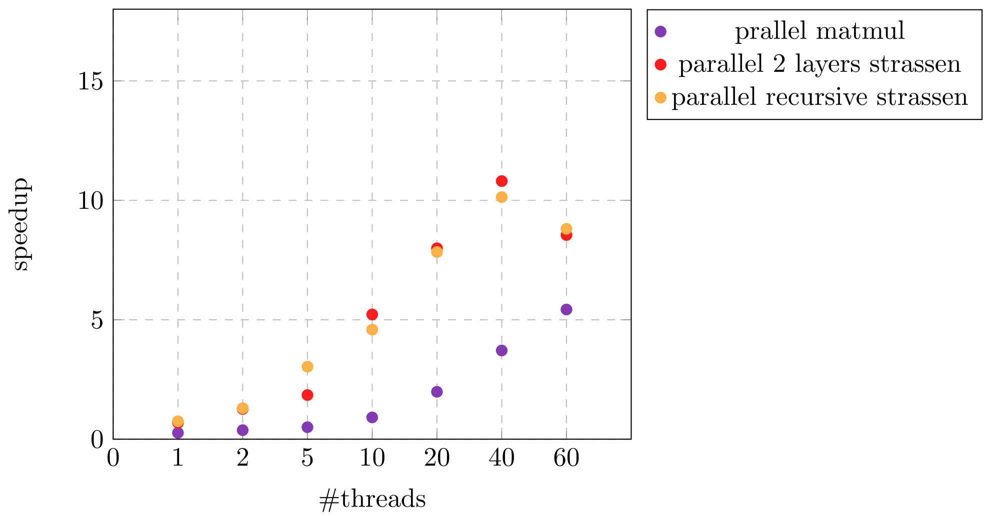
4.2 Comparison of parallel matrix multiplication and parallel Strassen's algorithm to the sequential Strassen's algorithm with different amount of threads



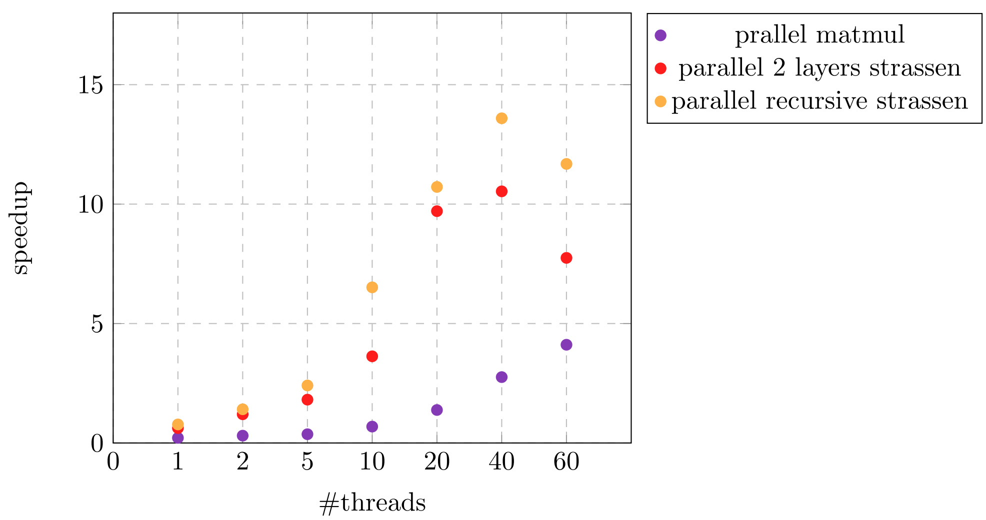




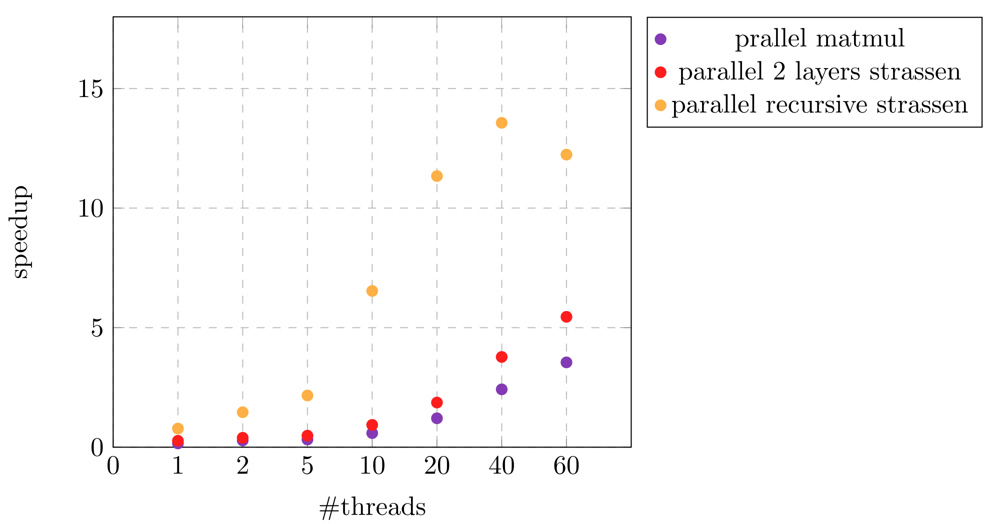
Matrices size 2048 × 2048



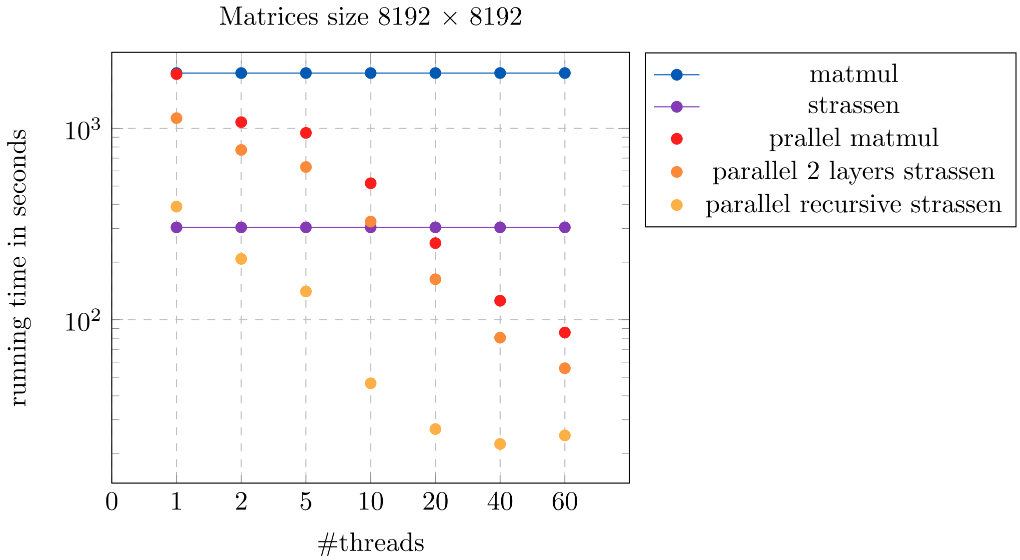
Matrices size 4096 × 4096



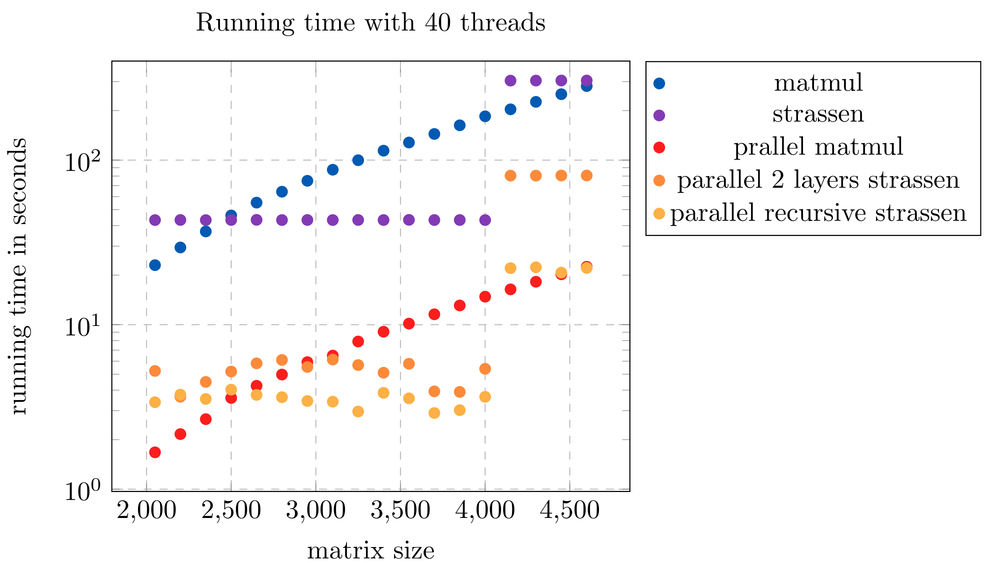
Matrices size 8192 × 8192

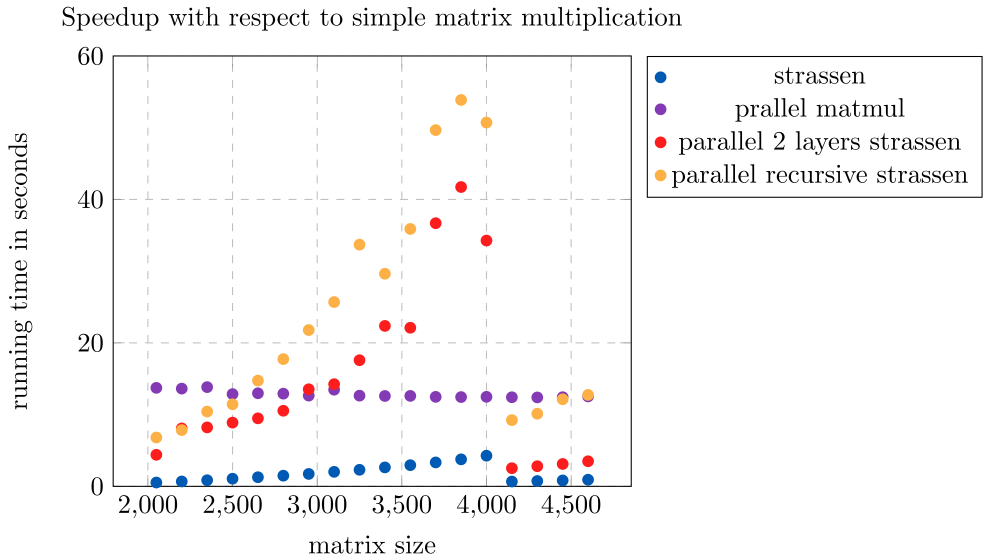


4.3 Running times for all implemeted algorithms and different numbers of threads



4.4 All implemeted algorithms for different sizes of matrices





5 Conclusion