Final Project
INF236: Parallel Programming
Parallel Matrix Multiplication

Kateřina Čížková, Luca Klingenberg

May 9, 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 Algorithms

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

2.1 Matrix Multiplication

Algorithm 1 Matrix Multiplication

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

```
1: function MATMUL($\mathbf{A}, \mathbf{B}$)
2:     for $i = 0, \ldots, n-1$ do
3:         for $j = 0, \ldots, n-1$ do
4:             $c[i][j] = 0$
5:         for $k = 0, \ldots, n-1$ do
6:             for $j = 0, \ldots, n-1$ do
7:                 $c[i][j] += a[i][k] \cdot b[k][j]$
8:     return C
```

2.2 Parallel Matrix Multiplication

2.3 Strassen Algorithm

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

$$A \cdot B = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \cdot \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} = \begin{pmatrix} A_{00}B_{00} + A_{01}B_{10} & A_{00}B_{01} + A_{01}B_{11} \\ A_{10}B_{00} + A_{11}B_{10} & A_{10}B_{01} + A_{01}B_{11} \end{pmatrix} = \begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix} = C$$

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. The algorithm is used recursively for these multiplications.

Algorithm 2 Strassen 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 $\mathbf{A} \cdot \mathbf{B}$
    $\mathbf{P}_1 = strassen(\mathbf{A}_{00} + \mathbf{A}_{11}, \mathbf{B}_{00} + \mathbf{B}_{11}, \frac{n}{2})$
    $\mathbf{P}_2 = strassen(\mathbf{A}_{10} + \mathbf{A}_{11}, \mathbf{B}_{00}, \frac{n}{2})$
    $\mathbf{P}_3 = strassen(\mathbf{A}_{00}, \mathbf{B}_{01} - \mathbf{B}_{11}, \frac{n}{2})$
    $\mathbf{P}_4 = strassen(\mathbf{A}_{11}, \mathbf{B}_{10} - \mathbf{B}_{00}, \frac{n}{2})$
    $\mathbf{P}_5 = strassen(\mathbf{A}_{00} + \mathbf{A}_{01}, \mathbf{B}_{11}, \frac{n}{2})$
    $\mathbf{P}_6 = strassen(\mathbf{A}_{10} - \mathbf{A}_{00}, \mathbf{B}_{00} + \mathbf{B}_{01}, \frac{n}{2})$
    $\mathbf{P}_7 = strassen(\mathbf{A}_{01} - \mathbf{A}_{11}, \mathbf{B}_{10} + \mathbf{B}_{11}, \frac{n}{2})$
    $\mathbf{C}_{00} = \mathbf{P}_1 + \mathbf{P}_4 - \mathbf{P}_5 + \mathbf{P}_7$
    $\mathbf{C}_{01} = \mathbf{P}_3 + \mathbf{P}_5$
    $\mathbf{C}_{10} = \mathbf{P}_2 + \mathbf{P}_4$
    $\mathbf{C}_{11} = \mathbf{P}_1 - \mathbf{P}_2 + \mathbf{P}_3 + \mathbf{P}_6$
    return $\mathbf{C}$
```

Strassen's algorithm is recursive and it is working with quarters of input matrices. Because of that, it is useful to use z-ordering for storing matrices in our implementation. Its advantage is that all submatrices are stored in consecutive parts of memory. But we switch from Strassen's algorithm to the simple matrix multiplication when the matrices are too small so the Strassen's algorithm overhead is too big. For the simple matrix multiplication is better to have the matrices ordered row-wise. We combined both together and used the z-ordering down to the cutting point of recursion. We left the small submatrices which are multiplied with the simple multiplication algorithm in row-wise order.

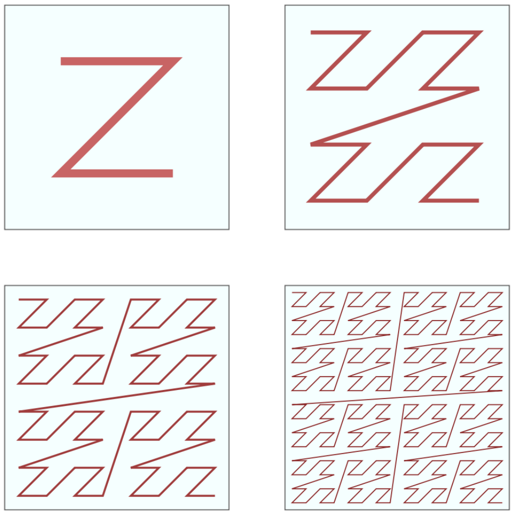


Figure 1: Z-layout

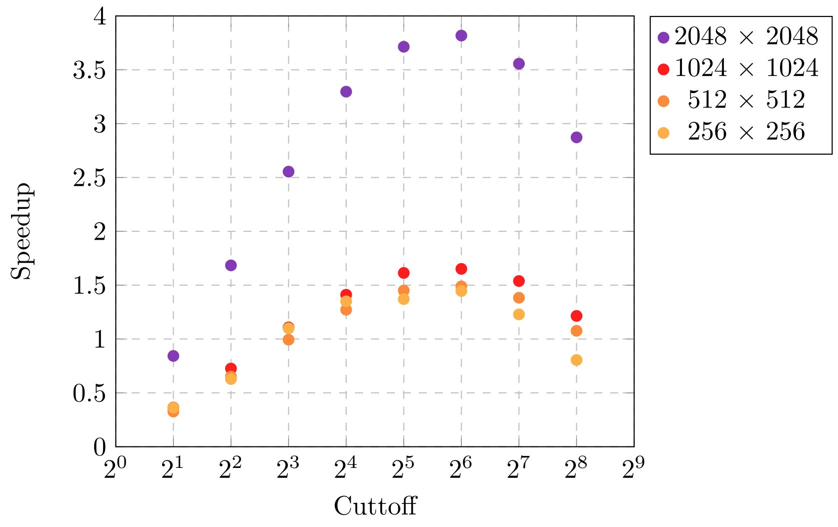
2.4 Parallel Matrix Multiplication

We just parallelized the outermost loop of the simple matrix multiplication algorithm described in section 2.1.

2.5 Parallel Strassen Algorithm

3 Experiments

Sequential Strassen Algorithm with different values for the cutoff level



4 Conclusion