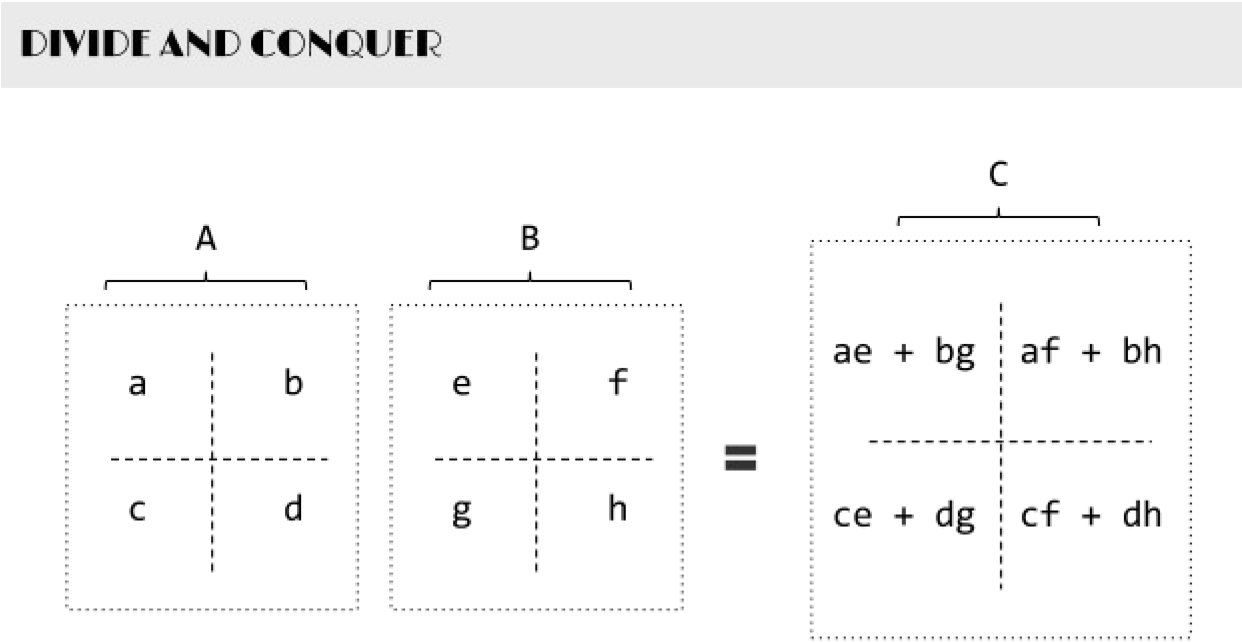
Optimization of Strassen’s Method for Matrix Multiply

Austin Schiller and Larry Sun

Intuitive Recursive Matrix Multiply

- Divide both the A and B matrices into a **4** sub matrices
- Treating the submatrices as elements, do matrix multiplication to get the resulting C matrix
- Repeat for all **8** new matrix multiplies (ae, bg, af… ect.)
- Algorithm runs at O(N^3)



- A, B, and C are of size NxN.
- a,b,c,d,e,f,g, and h submatrices of size N/2 x N/2

Strassen's Matrix Multiply

- Divide both the A and B matrices into a **4** sub matrices
- Create the **7** listed products using the submatrices (M1-M7)
- For all **7** submatrices M1-M7, do Strassen's Method on each submatrix!
- Once the submatrix is at a certain optimal size (size<=THRESHOLD), do regular matrix-matrix multiply
- Algorithm runs at O(N^2.8)

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

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

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

- A, B, and C are of size NxN and the size is a power of 2
- a11, a12, a21, a22 are submatrices of size N/2 x N/2. Same with the b11, b12...c22

Why Optimizing Strassen's Method is Difficult

- Memory management-
  - How to do all of the calculations with minimal extra space (allocating submatrices, temp matrices, etc.)
  - When to stop recursively creating matrices and use the serial method
  - The correct order of operation to avoid data dependencies
- Strassen-Winograd method
  - Use ⅔ N^2 extra space with specific scheduling to further optimize the result

Strassen-Winograd method scheduling

#	operation	loc.	#	operation	loc.
1	$Z_1 = C_{22} - C_{12}$	C_{22}	14	$P_2 = \mathtt{Acc}(\alpha A_{12} B_{21} + \beta C_{11})$	C_{11}
2	$Z_3 = C_{12} - C_{21}$	C_{12}	15	$\mathbf{U_1} = P_1 + P_2$	C_{11}
3	$S_1 = A_{21} + A_{22}$	X	16	$\mathbf{U_5} = U_2 + P_3$	C_{12}
4	$T_1 = B_{12} - B_{11}$	Y	17	$S_3 = A_{11} - A_{21}$	X
5	$P_5 = \mathtt{Acc}(\alpha S_1 T_1 + \beta Z_3)$	C_{12}	18	$T_3 = B_{22} - B_{12}$	Y
6	$S_2 = S_1 - A_{11}$	X	19	$U_3 = P_7 + U_2$	C_{21}
7	$T_2 = B_{22} - T_1$	Y		$= \alpha\mathtt{AcLR}(S_3 T_3 + U_2)$	
8	$P_6 = \mathtt{Acc}(\alpha S_2 T_2 + \beta C_{21})$	C_{21}	20	$\mathbf{U_7} = U_3 + W_1$	C_{22}
9	$S_4 = A_{12} - S_2$	X	21	$T_1' = B_{12} - B_{11}$	Y
10	$W_1 = P_5 + \beta Z_1$	C_{22}	22	$T_2' = B_{22} - T_1'$	Y
11	$P_3 = \mathtt{Acc}(\alpha S_4 B_{22} + P_5)$	C_{12}	23	$T_4 = T_2' - B_{21}$	Y
12	$P_1 = \alpha A_{11} B_{11}$	X	24	$\mathbf{U_6} = U_3 - P_4$	C_{21}
13	$U_2 = P_6 + P_1$	C_{21}		$= -\alpha\mathtt{AccR}(A_{22} T_4 - U_3)$	

Strassen Optimization: Saving Some Space

- Create new matrix add/sub functions that take in two **full** matrices and puts the result in a **smaller** submatrix
- **startrow, startcol** are passed in to specify which submatrix(a11, b21, etc.) of the **full** matrix are being worked on
- For example, if sublength=length(A)/2, then:

m_add(A, B, c11, 0, 0, sublength, sublength)

is the same as:

a11+b22=c11

- This saves unnecessary allocating of some submatrices (a11, a12, a21...)

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

```
void m_add(matrix_ptr a, matrix_ptr b, matrix_ptr c, int startrow1, int
startcol1, int startrow2, int startcol2)
{
        int i, j;
        int length = get_matrix_length(a);
        int sublength = get_matrix_length(c);
        data_t *m1 = get_matrix_start(a);
        data_t *m2 = get_matrix_start(b);
        data_t *m3 = get_matrix_start(c);

        for(i=0; i < sublength; i++){
                for(j=0; j < sublength; j++){
                        m3[i*sublength + j] = m1[(i+startrow1)*length +
startcol1 + j] + m2[(i+startrow2)*length + startcol2 + j];
                }
        }
}
```

Strassen's Method with Loop Unrolling

- Unroll the arithmetic operations (matrix addition/subtraction)
- Unroll the data transferring of the main A, B matrices into the submatrices for work
- Unrolling greater than 8 was found to be ineffective...

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

```
for(i=0; i < sublength; i++){
        for(j=0; j < sublength; j+=UNROLL_THRESHOLD){
                a11->data[i*sublength+j] = a0[i*length+j];
                b11->data[i*sublength+j] = b0[i*length+j];
                a22->data[i*sublength+j] = a0[(i+sublength)*length+sublength+j];
                b22->data[i*sublength+j] = b0[(i+sublength)*length+sublength+j];

                a11->data[i*sublength+j+1] = a0[i*length+j+1];
                b11->data[i*sublength+j+1] = b0[i*length+j+1];
                a22->data[i*sublength+j+1] = a0[(i+sublength)
*length+sublength+j+1];
                b22->data[i*sublength+j+1] = b0[(i+sublength)
*length+sublength+j+1];

                a11->data[i*sublength+j+2] = a0[i*length+j+2];
                b11->data[i*sublength+j+2] = b0[i*length+j+2];
                a22->data[i*sublength+j+2] = a0[(i+sublength)
*length+sublength+j+2];
                b22->data[i*sublength+j+2] = b0[(i+sublength)
*length+sublength+j+2];

                a11->data[i*sublength+j+3] = a0[i*length+j+3];
                b11->data[i*sublength+j+3] = b0[i*length+j+3];
                a22-[redacted]data[i*sublength+j+3] = a0[(i+sublength)
*length+sublength+j+3[redacted];
                b22->data[i*sublength+j+3] = b0[(i+sublength)
*length+sublength+j+3[redacted];
```

Strassen’s Method with OpenMP

- Since we need to create 8 new submatrices (a11, a12...c22), we can parallelize the copying of the A, B, and C matrices to the submatrices
- Every time an operation is done on across elements of a submatrix, begin the loops with **parallel for** statements
- Matrix addition and matrix subtraction is omp optimized

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

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

$$M_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$
$$M_2 = (A_{21} + A_{22}) \times B_{11}$$
$$M_3 = A_{11} \times (B_{12} + B_{22})$$
$$M_4 = A_{22} \times (B_{21} + B_{11})$$
$$M_5 = (A_{11} + A_{12}) \times B_{22}$$
$$M_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

```
#pragma omp parallel shared(a0,b0,a11,a22,b11,b22,sublength,chunk) private(i,
j) num_threads(OMP_THREADS)
{
#pragma omp for schedule (dynamic,chunk) nowait
for(i=0; i < sublength; ++i){
        for(j=0; j < sublength; ++j){
                a11->data[i*sublength+j] = a0[i*length+j];
                b11->data[i*sublength+j] = b0[i*length+j];
                a22->data[i*sublength+j] = a0[(i+sublength)
        *length+sublength+j];
                b22->data[i*sublength+j] = b0[(i+sublength)
        *length+sublength+j];
        }
}
}
```

⋮

main Strassen’s work...

⋮

```
#pragma omp parallel shared(c0,c11,c12,c21,c22,sublength,chunk) private(i,j)
num_threads(OMP_THREADS)
{
#pragma omp for schedule (dynamic,chunk) nowait
for(i=0; i<sublength; ++i){
        for(j=0; j<sublength; ++j){
                c0[i*length+j] = c11->data[i*sublength+j];
                c0[i*length+sublength+j] = c12->data[i*sublength+j];
                c0[(i+sublength)*length+j] = c21->data[i*sublength+j];
                c0[(i+sublength)*length+sublength+j] = c22->data
        [i*sublength+j];
        }
}
}
```

Strassen's Using Pthreads

- Create 7 threads, each thread works on one of the **M_i** matrices, where i ranges from 1 to 7
- Call the recursive function inside each of the 7 threads
- When a small matrix size is reached, finish with the original matrix multiply function
- When all threads are finished with their Mi, join them together in main function and do the arithmetic to get C

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

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

P1 $\quad M_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$

P2 $\quad M_2 = (A_{21} + A_{22}) \times B_{11}$

P3 $\quad M_3 = A_{11} \times (B_{12} + B_{22})$

P4 $\quad M_4 = A_{22} \times (B_{21} + B_{11})$

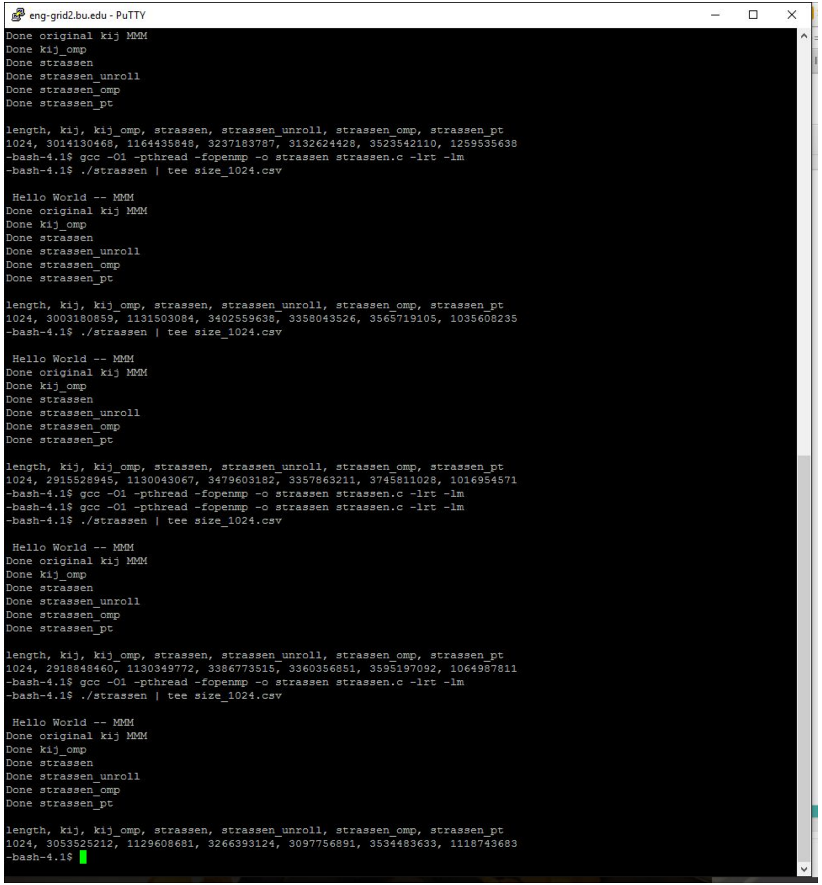
P5 $\quad M_5 = (A_{11} + A_{12}) \times B_{22}$

P6 $\quad M_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$

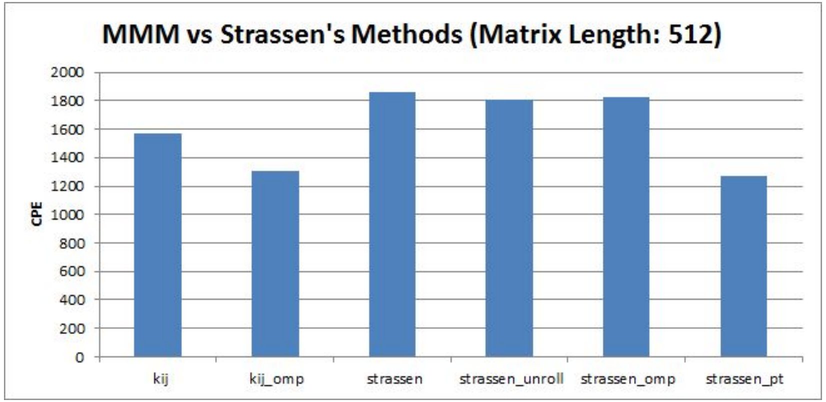
P7 $\quad M_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$

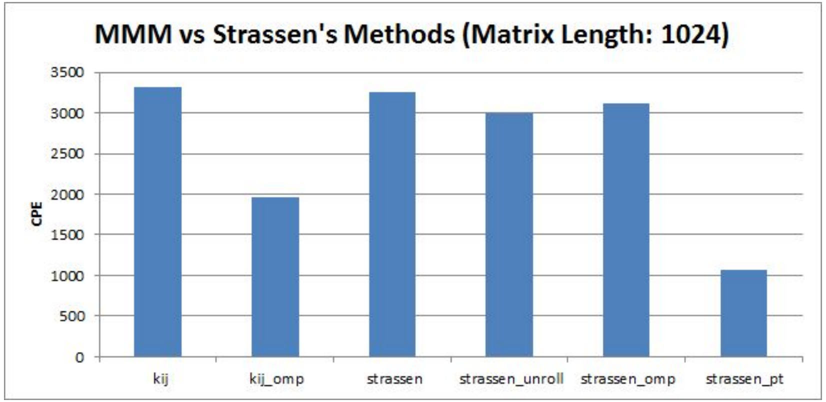
Experimental Procedure

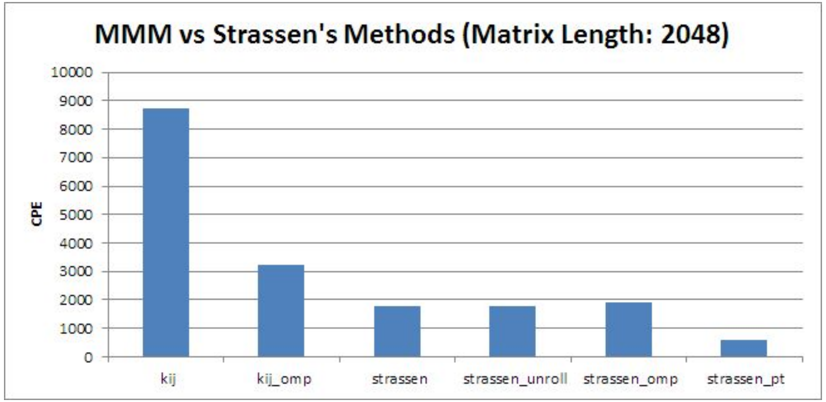
- Tested on eng-grid2
- Used -O1 optimization
- Since Strassen's Method performs worse than serial MMM at **small** matrix sizes (due to recursion/mem allocs), we tested with sizes: **512, 1024, 2048, 4096**
- In the Strassen versions, the optimal threshold for starting normal MMM is different at different matrix sizes.
(On average: Threshold = length/8 or 16)

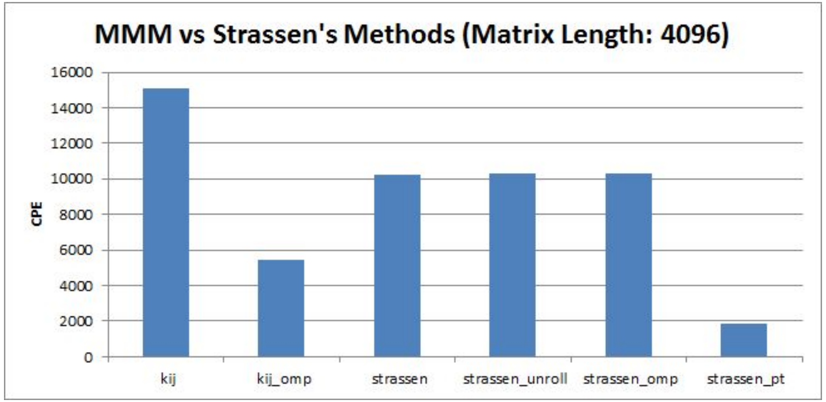


Results


MMM vs Strassen's Methods (Matrix Length: 512)
CPE
0
200
400
600
800
1000
1200
1400
1600
1800
2000
kij
kij_omp
strassen
strassen_unroll
strassen_omp
strassen_pt


MMM vs Strassen's Methods (Matrix Length: 1024)
CPE
0
500
1000
1500
2000
2500
3000
3500
kij
kij_omp
strassen
strassen_unroll
strassen_omp
strassen_pt


MMM vs Strassen's Methods (Matrix Length: 2048)
CPE
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
kij
kij_omp
strassen
strassen_unroll
strassen_omp
strassen_pt


MMM vs Strassen's Methods (Matrix Length: 4096)
CPE
0
2000
4000
6000
8000
10000
12000
14000
16000
kij
kij_omp
strassen
strassen_unroll
strassen_omp
strassen_pt

Conclusion

- At large matrix sizes, the Strassen methods perform significantly better
- Strassen omp and unrolled versions performs about the same as normal version
- Strassen with Pthreads performs very well! (At 4096, almost a factor of 8x speedup)
- MMM_omp still performs better than most Strassen versions, but at the cost of some error (~1%) due to race conditions

Possible Improvements in the Future

- Perform more fine-grained omp sections in the Strassen version (perhaps assign the 7 matrices to individual threads like in pthreads)
- Unroll the recursion and implement iteration to allow for more parallelism
- Implement Strassen’s Method with Intel Intrinsics (perform vector arithmetic with the smaller submatrices?)
- Maybe implement blocking
- Combine the best methods

Thank You!