Mátrix szorzás gyorsítása

A mátrixszorzás gyorsítása C-ben többféle módon lehetséges. Az alábbiakban bemutatunk néhány optimalizálási technikát:

Optimális sorrend

A hagyományos mátrixszorzás három beágyazott ciklusból áll (i, j, k sorrendben). Azonban a memóriaelérés optimalizálásával (pl. oszlopok helyett sorok szerint haladás) jelentősen növelhetjük a sebességet.

```
void multiplyMatrices(int **A, int **B, int **C, int N) {
    for (int i = 0; i < N; i++) {
        for (int k = 0; k < N; k++) {  // Megváltoztatott sorrend
            int r = A[i][k];
            for (int j = 0; j < N; j++) {
                C[i][j] += r * B[k][j];  // Csökkentett memóriahozzáférés
            }
        }
    }
}
```

Miért gyorsabb?

- Az **A** mátrix sorait gyorsabban beolvassuk a CPU gyorsítótárba (cache).
- Az előre kiszámított **r** változó csökkenti az A[i][k] memória-hozzáférését.

Blokkosítás

A blokkosított mátrixszorzás során a mátrixokat kisebb blokkokra bontjuk, és ezeket a blokkokat külön-külön számoljuk ki, így kihasználva a CPU gyorsítótárát.

$$ A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

$$ B = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

A fenti mátrixokat 2×2-es blokkokra bontjuk, tehát így:

$$ A = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} & \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix} \\ \begin{bmatrix} 9 & 10 \\ 13 & 14 \end{bmatrix} & \begin{bmatrix} 11 & 12 \\ 15 & 16 \end{bmatrix} \end{bmatrix}$$

$$ B = \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0

\end{bmatrix} \end{bmatrix}$$