

Analiza równoległego mnożenia macierzy

Maciej Gawrysiak

Specyfikacja

CPU: AMD Ryzen™ 7 7735U with Radeon™ Graphics × 16

RAM: 16.0 GiB

OS: Fedora Linux 41 (Workstation Edition)

Kompilator: clang version 19.1.7

Zrównoleglone czytanie z dysku

```
Matrix m1, m2;
#pragma omp parallel sections
{
    #pragma omp section
    {
        m1 = matrix_read(in1);
    }
    #pragma omp section
    {
        m2 = matrix_read(in2);
    }
}
```

Zrównoleglenia pętli

Zrównoleglenie po trzeciej pętli zwraca niepoprawne wyniki. Zrównoleglenie trzeciej pętli w algorytmie prowadzi do nieprawidłowych wyników z powodu race conditions. Wiele wątków jednocześnie próbuje aktualizować ten sam element macierzy wynikowej `m.items` bez synchronizacji, powodując race conditions, a tym samym nieprawidłowe wyniki.

- `MATRIX_PARALLELIZE_FIRST` i `MATRIX_PARALLELIZE_SECOND` zrównoleglają odpowiednio zewnętrzną i środkową pętlę, które rozdzielają pracę między wątki bez race conditions.
- `MATRIX_PARALLELIZE_THIRD` zrównolegla najbardziej wewnętrzną pętlę, co prowadzi do jednoczesnej aktualizacji tego samego elementu `m.items[i * m.cols + j]` przez wiele wątków.

Tabela czasów

Mierzenie czasu (main.cpp)

```
double start = omp_get_wtime();
const Matrix m3 = matrix_multiply(&m1, &m2);
std::cout << "Multiplication time: " << omp_get_wtime() - start << "s\n";
```

Zrównoleglona pierwsza pętla (nieznacznie wydajniejsza)

Rozmiar	Sek.	2	4	8	16
100	0.00464511	0.00305486	0.00205779	0.00184894	0.00247097
500	0.429552	0.228534	0.119312	0.10316	0.0613549
1000	3.50236	1.79631	0.946576	0.53452	0.523507
2000	34.7186	18.0619	9.39324	5.21831	4.90127

matrix.cpp

```
#ifdef MATRIX_PARALLELIZE_FIRST
#pragma omp parallel for
#endif
for (uint16_t i = 0; i < m1->rows; ++i) {
    for (uint16_t j = 0; j < m2->cols; ++j) {
        for (uint16_t k = 0; k < m1->cols; ++k) {
            m.items[i * m.cols + j] +=
                m1->items[i * m1->cols + k] * m2->items[k * m2->cols + j];
        }
    }
}
```

Zrównoleglona druga pętla

Rozmiar	Sek.	2	4	8	16
100	0.004884	0.00382996	0.00228691	0.00237513	0.00300694
500	0.431423	0.22843	0.121885	0.066236	0.115295
1000	3.56473	1.86065	0.974723	0.628134	0.656047
2000	34.8433	17.8441	9.41069	5.38997	6.14923

matrix.cpp

```
for (uint16_t i = 0; i < m1->rows; ++i) {
#ifdef MATRIX_PARALLELIZE_SECOND
#pragma omp parallel for
#endif
    for (uint16_t j = 0; j < m2->cols; ++j) {
        for (uint16_t k = 0; k < m1->cols; ++k) {
            m.items[i * m.cols + j] +=
                m1->items[i * m1->cols + k] * m2->items[k * m2->cols + j];
        }
    }
}
```

```

    }
  }
}

```

Badania dla zrównoleglonej pierwszej pętli

Schedule static

Fragment	Rozmiar	8	16
10	1000	0.586705	0.55614
10	2000	5.10526	5.20989
50	1000	0.623414	0.703368
50	2000	5.11087	4.86923
100	1000	0.802295	0.728099
100	2000	6.03135	5.7758

matrix.cpp

```

#ifdef MATRIX_PARALLELIZE_FIRST
#pragma omp parallel for schedule(static, 100)
#endif
for (uint16_t i = 0; i < m1->rows; ++i) {
    for (uint16_t j = 0; j < m2->cols; ++j) {
        for (uint16_t k = 0; k < m1->cols; ++k) {
            m.items[i * m.cols + j] +=
                m1->items[i * m1->cols + k] * m2->items[k * m2->cols + j];
        }
    }
}
}

```

Schedule dynamic

Fragment	Rozmiar	8	16
10	1000	0.598091	0.530066
10	2000	5.92392	4.86848
50	1000	0.621086	0.706843
50	2000	5.19588	5.04123
100	1000	0.843801	0.725773
100	2000	6.08794	5.79966

matrix.cpp

```

#ifdef MATRIX_PARALLELIZE_FIRST
#pragma omp parallel for schedule(dynamic, 100)
#endif

```

```

for (uint16_t i = 0; i < m1->rows; ++i) {
    for (uint16_t j = 0; j < m2->cols; ++j) {
        for (uint16_t k = 0; k < m1->cols; ++k) {
            m.items[i * m.cols + j] +=
                m1->items[i * m1->cols + k] * m2->items[k * m2->cols + j];
        }
    }
}

```

Schedule guided

Fragment	Rozmiar	8	16
10	1000	0.590611	0.543017
10	2000	5.19615	4.83037
50	1000	0.621673	0.660162
50	2000	5.46533	5.16736
100	1000	0.815689	0.730184
100	2000	6.09648	5.87968

matrix.cpp

```

#ifdef MATRIX_PARALLELIZE_FIRST
#pragma omp parallel for schedule(guided, 100)
#endif
for (uint16_t i = 0; i < m1->rows; ++i) {
    for (uint16_t j = 0; j < m2->cols; ++j) {
        for (uint16_t k = 0; k < m1->cols; ++k) {
            m.items[i * m.cols + j] +=
                m1->items[i * m1->cols + k] * m2->items[k * m2->cols + j];
        }
    }
}

```

Analiza zależności

Dla pólki

normalna dla tablicy B, dla B ujemna tensor

$$\begin{aligned} \text{for } i \in 0; i < m; ++i \\ \text{for } j \in 0; j < m; ++j \\ \text{for } k \in 0; k < m; ++k \\ C[i][j][k] = C[i][j][k] + A[i][k] - B[k][j] \end{aligned}$$

$$0, 0, 0 \quad C(0,0) = C(0,0) + A(0,0) - B(0,0)$$

$$0, 0, 1 \quad C(0,0) = C(0,0) + A(0,1) - B(1,0)$$

$$0, 0, 2 \quad C(0,0) = C(0,0) + A(0,2) - B(2,0)$$

$$K = J - 1$$

$$I = J - K$$

$$I = 0$$

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \bar{A} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \bar{B} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$A + \bar{A} = B + \bar{B}$$

$$A(K = A) + \bar{A} = B + \bar{B}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} k_i \\ k_j \\ k_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \\ k \end{bmatrix} + 0 - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \\ k \end{bmatrix} = 0$$

$$\begin{bmatrix} k_i \\ k_j \end{bmatrix} = \begin{bmatrix} i \\ j \end{bmatrix} \Rightarrow \begin{cases} k_i = i \\ k_j = j \end{cases}$$

Zrównoleglenie jest możliwe ze względu na brak zależności, jednakże należy wziąć pod uwagę instrukcje

$$\Delta i = i' - i \Rightarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Delta k = k' - k \Rightarrow \Delta k = k' - k = 1 \neq 0$$

$$K = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Zrównoleglenie jest możliwe jedynie na poziomie i, dlatego pólki i przypadek pólki tenże zrównoleglenie odpowiednio do same conditions.