Programowanie równoległe
Laboratorium 7

Sprawozdanie

Zadanie 2

Zadanie brzmi «*Napisanie programu mnożącego 2 macierze o wymiarach NxM i MxP.*»

Kod źródłowy rozwiązania:

```
#include <stdio.h>
#include <omp.h>
#include <math.h>

void fillMatrix(size_t n, size_t m, int matrix[n][m], short
number) {
    for (short i = 0; i < n; ++i) {
      for (short j = 0; j < m; ++j) {
           matrix[i][j] = number;
      }
    }
}

void multiplyMatrices(size_t n, size_t m, size_t p, int
matrix1[n][m], int matrix2[m][p], int result[n][p]) {
    for (short i = 0; i < n; ++i) {
      for (short j = 0; j < p; ++j) {
           for (short k = 0; k < m; ++k) {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
           }
      }
    }
}

// void printMatrix(size_t n, size_t m, int matrix[n][m]) {
//     for (short i = 0; i < n; ++i) {
//          for (short j = 0; j < m; ++j)
//               printf("%d\t", matrix[i][j]);

//          printf("\n");
//     }
// }

int main() {
```

```
    for (short i = 0; i < 3; ++i) {
      double start, end;
      short n = 2 * pow(10, i);
      short m = 3 * pow(10, i);
      short p = 4 * pow(10, i);
      int matrix1[n][m], matrix2[m][p], matrix3[n][p];

      printf("Multiplying %dx%d & %dx%d matrices\n", n, m, m,
p);

      fillMatrix(n, m, matrix1, 2);
      fillMatrix(m, p, matrix2, 2);
      fillMatrix(n, p, matrix3, 0);

      start = omp_get_wtime();
      multiplyMatrices(n, m, p, matrix1, matrix2, matrix3);
      end = omp_get_wtime();

      printf("Time taken:\t%f\n", end - start);

      // printMatrix(n, p, matrix3);
    }

    return 0;
}
```

Wydruk programu:

```
Multiplying 2x3 & 3x4 matrices
Time taken:    0.000000
Multiplying 20x30 & 30x40 matrices
Time taken:    0.000073
Multiplying 200x300 & 300x400 matrices
Time taken:    0.08818
```

Zadanie 4

Zadanie brzmi «*Zrównoleglenie napisanego programu, przetestowanie dla różnych strategii podziału i metod zrównoleglenia (pętla wewnętrzna/zewnętrzna).*»

Kod źródłowy rozwiązania:

```
#include <stdio.h>
#include <omp.h>
```

```
#include <math.h>

void fillMatrix(size_t n, size_t m, int matrix[n][m], short
number) {
    for (short i = 0; i < n; ++i) {
      for (short j = 0; j < m; ++j) {
           matrix[i][j] = number;
      }
    }
}

void multiplyMatrices(size_t n, size_t m, size_t p, int
matrix1[n][m], int matrix2[m][p], int result[n][p]) {
    #pragma omp parallel for shared(matrix1, matrix2, result)
num_threads(8) schedule(static, 3)
    for (short i = 0; i < n; ++i) {
      for (short j = 0; j < p; ++j) {
           for (short k = 0; k < m; ++k) {
                int tmp = matrix1[i][k] * matrix2[k][j];

                #pragma omp critical
                {
                     result[i][j] += tmp;
                }
           }
      }
    }
}

// void printMatrix(size_t n, size_t m, int matrix[n][m]) {
//     for (short i = 0; i < n; ++i) {
//          for (short j = 0; j < m; ++j)
//               printf("%d\t", matrix[i][j]);

//          printf("\n");
//     }
// }

int main() {
    for (short i = 0; i < 3; ++i) {
      double start, end;
      short n = 2 * pow(10, i);
      short m = 3 * pow(10, i);
      short p = 4 * pow(10, i);
      int matrix1[n][m], matrix2[m][p], matrix3[n][p];
```

```
      printf("Multiplying %dx%d & %dx%d matrices\n", n, m, m,
p);

      fillMatrix(n, m, matrix1, 2);
      fillMatrix(m, p, matrix2, 2);
      fillMatrix(n, p, matrix3, 0);

      start = omp_get_wtime();
      multiplyMatrices(n, m, p, matrix1, matrix2, matrix3);
      end = omp_get_wtime();

      printf("Time taken:\t%f\n", end - start);

      // printMatrix(n, p, matrix3);
    }

    return 0;
}
```

Zabezpieczenie zmiennej w funkcji multiplyMatrices() jest niezbędne dlatego że jeśli pętla wewnętrzna jest zrównoleglona, różne wątki nadpisują dane w komórce result[i][j]. Skorzystałem ze sposobu zabezpieczenia critical dlatego że, jak pokazały wyniki poprzedniego laboratorium, jest on najbardziej wydajny.

Wydruk programu:

```
Multiplying 2x3 & 3x4 matrices
Time taken:    0.001057
Multiplying 20x30 & 30x40 matrices
Time taken:    0.002178
Multiplying 200x300 & 300x400 matrices
Time taken:    2.482000
```

Zadanie 5

Zadanie brzmi «*Porównanie i analiza otrzymanych wyników*»

Wyniki różnych pomiarów:

threads	1	8	8	8	8	8	8	8	8
cycle	-	inner	outer	inner	outer	inner	outer	inner	outer

scheduling type	-	dynamic	dynamic	dynamic	dynamic	static	static	static	static
chunk	-	auto	auto	3	3	auto	auto	3	3
2x3 & 3x4	0.00000	0.00252	0.00296	0.00287	0.00061	0.00296	0.00326	0.00366	0.00106
20x30 & 30x40	0.00007	0.00478	0.00289	0.00394	0.00242	0.00313	0.00216	0.00323	0.00218
200x300 & 300x400	0.08818	4.76222	3.01953	3.29534	2.86909	2.86826	2.22477	3.05373	2.48200

Z wyników wynika że:

- Wykorzystanie jednego wątku w tym zadaniu jest najbardziej skuteczne. Najprawdopodobniej jest to tak dlatego że tworzenie dużej liczby wątków wymaga większej ilości zasobów niż obliczenia i/lub zabezpieczenie zmiennej potrzebuje dużo czasu.
- Zrównoleglenie pętli zewnętrznej jest bardziej wydajne, bo tworzy się mniejsza ilość wątków i zabezpieczenie zmiennej nie jest niezbędne.
- Zrównoleglenie typu static jest bardziej wydajne dla większych macierzy, a dynamic — dla mniejszych.

Mnożenie za pomocą programu macierzy A przez macierz B daje w wyniku macierz C, co udowadnia poprawność wyników.

A:

2	2	2
2	2	2

B:

2	2	2	2
2	2	2	2
2	2	2	2

C:

12	12	12	12
12	12	12	12