Sprawozdanie z Laboratorium Projektowania Systemów Informatycznych

Zadanie nr 1

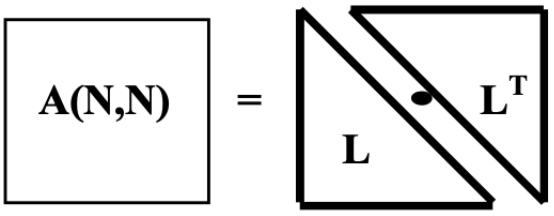
Tomasz Lech
Krzysztof Niemiec

Informatyka
specjalność PKiSI
Semestr V

Koszalin, 20.10.2023

Wprowadzenie

Celem ćwiczenia była implementacja algorytmu algebry liniowej w wybranym języku programowania oraz zmierzenie czasu potrzebnego na wykonanie owego algorytmu na CPU. Algorytmem realizowanym przez naszą grupę laboratoryjną był algorytm rozkładu macierzy metodą Cholesky'ego.



Rys 1. Wizualizacja działania dekompozycji Cholesky'ego.

Wynikiem rozkładu Cholesky'ego wykonanego na macierzy A jest macierz trójkątna dolna L, która spełnia następującą zależność:

$$A = LL^T \tag{1}$$

Aby rozkład Cholesky'ego był możliwy, macierz wejściowa A musi być *symetryczna* oraz *pozytywnie zdefiniowana* (ang. *symmetric positive definite,* w skrócie macierz SPD).

Razem z poleceniem do zadania otrzymaliśmy pseudokod realizujący rozkład Cholesky'ego:

Listing 1. Pseudokod dla rozkładu Cholesky'ego

```
for i := 1 to N do
begin
a_ii := SQRT(a_ii);
   for j := i+1 to N do
      a_ji := a_ji / a_ii ;
   for j := i+1 to N do
      for k := i+1 to j do
         a_jk := a_jk - a_ji * a_ki ;
end;
```

Realizacja zadania

Rozkład Cholesky'ego zrealizowaliśmy w języku C++. Pseudokod algorytmu przepisany do składni C++ jest pokazany na listingu 2.

Listing 2. Kod rozkładu Cholesky'ego w C++

```
typedef std::vector<float> vf;
typedef std::vector<vf> vvf;

vvf cholesky(vvf a, int n) {
    for(int i = 0; i < n; i++) {

        a[i][i] = sqrt(a[i][i]);

        for(int j = i+1; j < n; j++) {
            a[j][i] = a[j][i]/a[i][i];
        }

        for(int j = i+1; j < n; j++) {
            for(int k = i+1; k <= j; k++) {
                a[j][k] = a[j][k] - a[j][i]*a[k][i];
            }
        }
    }

    vvf l(n, vf(n));
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            if(i >= j) {
                l[i][j] = a[i][j];
            }
        }
    }
    return l;
}
```

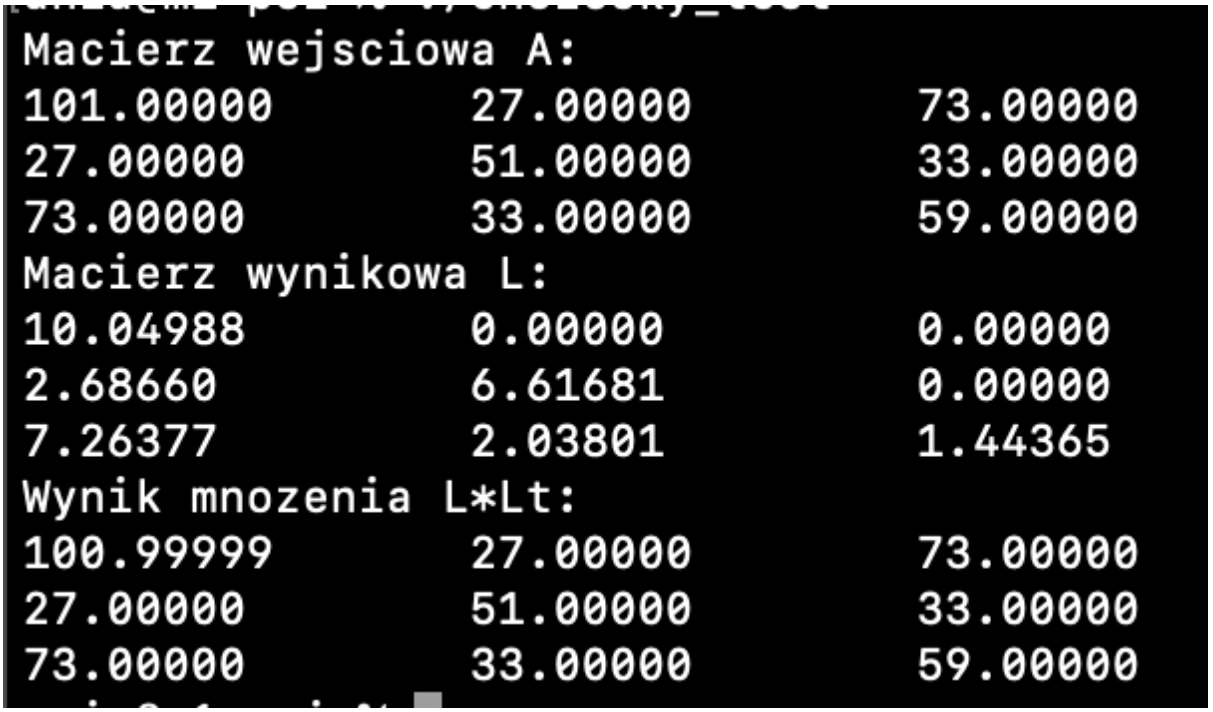
Do generowania danych testowych również wykorzystaliśmy język C++ oraz zależność, że dowolna macierz przemnożona przez jej macierz odwrotną jest macierzą SPD[1]. Kod generujący losowe macierze testowe jest pokazany na listingu 3.

[1] https://stackoverflow.com/questions/48736724/generate-matrix-symmetric-and-positive-definite

Listing 3. Kod generujący losową macierz SPD

```
#define RAND_A -10
#define RAND_B 10
typedef std::vector<float> vf;
typedef std::vector<vf> vvf;

int rand_range() {
    return (RAND_A)+std::rand()%((RAND_B)-(RAND_A));
}

vvf matmul_nxn(vvf a, vvf b, int n) {
    vvf res(n, vf(n));
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            for(int k = 0; k < n; k++) {
                res[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return res;
}

vvf inverse(vvf a, int n) {
    vvf res(n, vf(n));
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            res[i][j] = a[j][i];
        }
    }
    return res;
}

vvf random_spd_matrix(int n) {
    vvf res;
    for(int i = 0; i < n; i++) {
        vf row(n);
        std::generate(row.begin(), row.end(), rand_range);
        res.push_back(row);
    }

    vvf res_inv = inverse(res, n);
    res = matmul_nxn(res, res_inv, n);
    return res;
}
```

Ze względu na zależność (1), poprawność algorytmu można zweryfikować poprzez mnożenie macierzy L oraz L^T. Poprawność ta została empirycznie sprawdzona na niewielkich instancjach problemu.



Rys 2. Sprawdzenie poprawności algorytmu. Wartość 100.99999 w macierzy LL^T wynika z błędu użytego typu `float`.

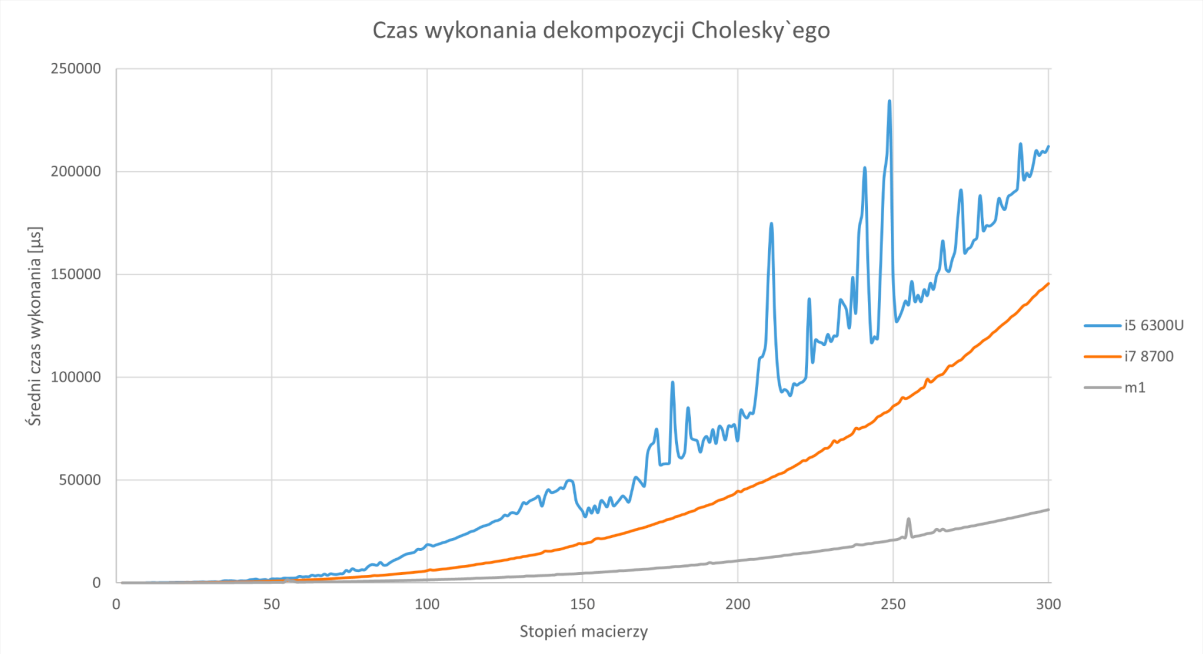
W celach pomiaru prędkości wykonania algorytmu, zmierzyliśmy czas wykonania funkcji `cholesky()` na trzech różnych urządzeniach. Badane procesory to:

- Intel Core i5-6300U
- Intel Core i7-8700
- Apple M1

Algorytm sprawdzono na macierzach o stopniu od 2 do 300. Dla każdego stopnia macierzy wykonano 10 pomiarów i policzono ich średnią. Czas wykonania algorytmu zmierzono w mikrosekundach przy pomocy biblioteki `chrono` języka C++. Funkcję `main()` przedstawiono na listingu 4. Porównanie czasów na powyższych procesorach przedstawiono na wykresie na rysunku 3.

Listing 4. Funkcja `main()`

```
typedef std::vector<float> vf;
typedef std::vector<vf> vvf;

int main(int argc, char* argv[]) {
    std::srand(std::time(nullptr));

    for(int n = 2; n <= 300; n++) {
        for(int i = 0; i < 10; i++ ){
            vvf a = random_spd_matrix(n);
            auto start =
std::chrono::high_resolution_clock::now();
            vvf l = cholesky(a, n);
            auto stop = std::chrono::high_resolution_clock::now();
            auto duration =
std::chrono::duration_cast<std::chrono::microseconds>(stop -
start);
            std::cout << duration.count() << " ";
        }
        std::cout << std::endl;
    }
    return 0;
}
```



Rys 3. Porównanie czasu wykonania funkcji `cholesky()` w zależności od stopnia macierzy wejściowej na badanych procesorach.

Podsumowanie

Pseudokod został poprawnie zaimplementowany w języku C++. Czas wykonania algorytmu został zmierzony oraz porównany między trzema popularnymi procesorami. Wykres odzwierciedla wielomianową złożoność algorytmu w każdym przypadku.

Interesującym zjawiskiem jest pojawienie się nieprecyzyjności w pomiarach dla większych wartości stopnia macierzy. Potencjalnym wytłumaczeniem jest fakt, że pomiar był różnicą między czasem wywołania funkcji a czasem wyjścia z funkcji - w tym czasie system operacyjny mógł potencjalnie wywłaszczyć wątek na swoje cele, co nie jest uwzględnione w pomiarze. Takie wywłaszczenie ma większą szansę zajścia w przypadku dłuższego wykonywania się funkcji, więc zachodzi częściej dla większego stopnia macierzy.

Koszalin, 20.10.2023