



Politechnika Krakowska im. Tadeusza Kościuszki
Wydział Informatyki i Telekomunikacji

Zrównanie algorytmu znajdowania największego elementu ze zbioru liczb

14 stycznia 2022

Tomasz Grzesik, Piotr Ksel

1 Wprowadzenie

Celem projektu jest przedstawienie różnic w czasie weryfikacji, która liczba w zbiorze jest największa. Zostanie omówiony model z wymianą komunikatów oraz model wirtualnej pamięci wspólnej pamięci wspólnej. Na końcu zaimplementowany został model hybrydowy.

1.1 Treść zadania

Znajdowanie k-tego największego elementu ze zbioru liczb.

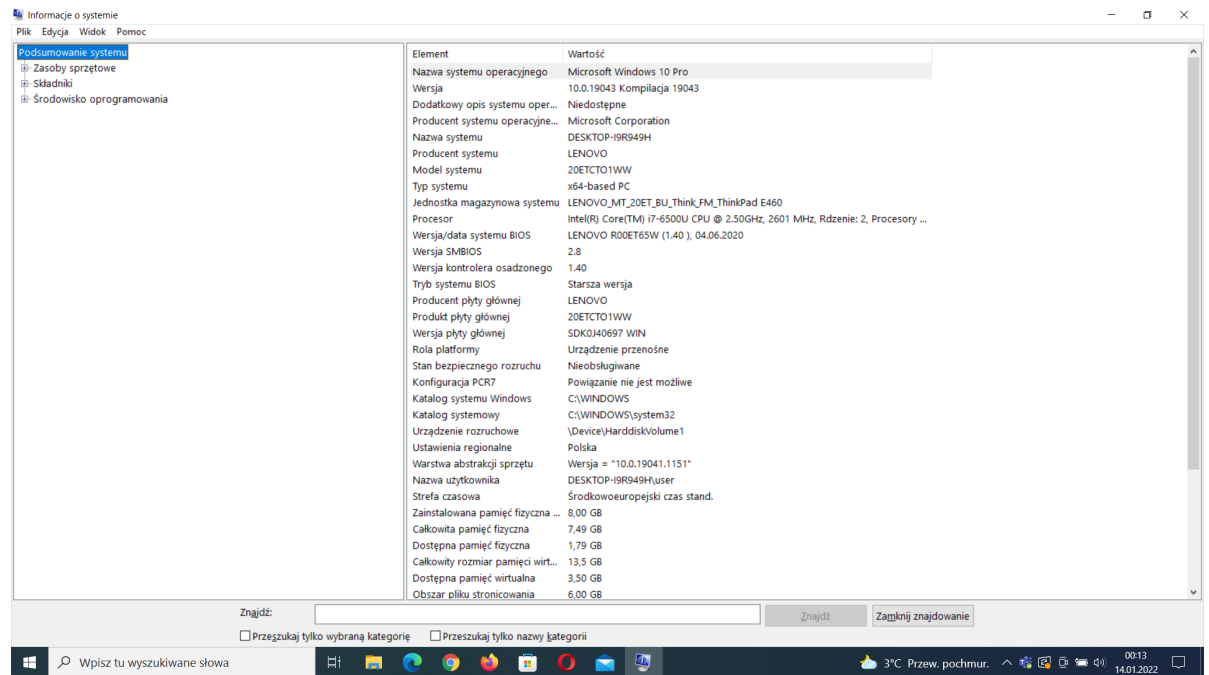
1.2 Dane wejściowe

Każde z wersji algorytmu posiada dane wejściowe, które są losowane generatorem liczb pseudolosowych. Poniżej funkcja odpowiedzialna za generowanie liczb.

```
for (int i(0); i < silk; i++)  
    nums.push_back(rand() % 10000);
```

1.3 Parametry maszyny wykonującej obliczenia

Na załączonym obrazku są poszczególne informacje o parametrze maszyny wykorzystanej do obliczeń.



2 Wersja w MPI

Model z wymianą komunikatów (Message Passing Interface) charakteryzuje się podziałem problemu na podproblemy, które są opracowywane przez odrębne procesy. Podproblemami są podzbiory wejściowego zbioru danych. Zbiór wejściowy został podzielony na niezależne fragmenty, dla których niezależnie się oblicza k -tą największą wartość. W procesie zerowym zostały wygenerowane dane wejściowe i zapisane do tablicy. W modelu z wymianą komunikatów dane wymieniane pomiędzy procesami przesyłane są za pomocą komunikatów. Implementacja programu w MPI wykorzystuje przysyłanie komunikatów typu jeden do jeden. Dzięki zastosowaniu `MPISend()` oraz `MPIRecv()` referencja do wejściowego zbioru danych oraz wartości k były przesyłane pomiędzy procesami dzięki temu można było wyznaczyć k -tą największą wartość.

```

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Request req1;
MPI_Status status;
int* tabAll = new int{ size };
if (my_rank == 0) {
    startParallel = MPI_Wtime();
    for (d = 0; d < size; d++) {
        if (d != my_rank) {
            int a = 4;
            MPI_Send(&a, 1, MPI_INT, d, 13, MPI_COMM_WORLD);
        }
    }
}
else if (my_rank != 0) {
    int b;
    MPI_Recv(&b, 1, MPI_INT, 0, 13, MPI_COMM_WORLD, &status);
}
reszta = MAX % size;
if (reszta == 0) wartosc = MAX / size;
else {
    if (my_rank == 0) wartosc = MAX;
    else wartosc = 0;
    if (my_rank == 0) {
        cout << "Nie można podzielić tablicy na %d " << size << "równych części!";
        cout << "K-ta wartość zostanie wyliczona przez 1 proces.";
    }
}
y = wartosc * my_rank;
z = wartosc * my_rank + wartosc;

```

3 Wersja w OpenMP

Model pamięci wspólnej algorytmu do wyznaczenia k-tej największej wartości w standardzie OpenMP (Open Multi-Processing). Standard ten wykorzystuje pracę na wielu wątkach oraz pamięć współdzieloną. Kod algorytmu został napisany w języku C++. Implementacja algorytmu odróżnia się od MPI brakiem przesyłania komunikatów, ponieważ wykorzystana została pamięć wspólna. Kod jest tożsamy z wersją sekwencyjną, jednakże pętla algorytmu została zrównoleglona. Kod zrównoleglenia algorytmu przedstawia rysunek poniżej.

```

vector<int> nums;
int k = 3;
int th = 10;
omp_set_num_threads(th);
int n = 50000;
int i;
for (int i = 0; i < n; i++)
    nums.push_back(rand() % 100000);
double startParallel = 0.0, stopParallel = 0.0;
startParallel = omp_get_wtime();
#pragma omp parallel for num_threads(th)
for (int i = 0; i < th; i++)
    KthLargest2(nums, k);
stopParallel = omp_get_wtime();
cout << "Array: ";
cout << k << " Largest:" << KthLargest2(nums, k) << endl;
double Diff = stopParallel - startParallel;
cout << "Execution time: " << Diff;

```

4 Wersja hybrydowa

Implementacja hybrydowa łączy oba modele programowania równoległego: model z wymianą komunikatów i model pamiędzi wspólnej. Kod charakteryzuje się dwoma poziomami zrównoleglenia. Kod programu początkowo rozdziela zadania na poszczególne procesy za pomocą funkcji MPI, natomiast w obrębie sprawdzenia warunku uruchamiane są dyrektywy OpenMP.

```

if (my_rank == 0) {
    startParallel = MPI_Wtime();
    for (d = 0; d < size; d++) {
        if (d != my_rank) {
            int a = 4;
            MPI_Send(&a, 1, MPI_INT, d, 13, MPI_COMM_WORLD);
        }
    }
}
else if (my_rank != 0) {
    int b;
    MPI_Recv(&b, 1, MPI_INT, 0, 13, MPI_COMM_WORLD, &status);
}

reszta = MAX % size;
if (reszta == 0) wartosc = MAX / size;
else {
    if (my_rank == 0) wartosc = MAX;
    else wartosc = 0;
    if (my_rank == 0) {
        cout << "Nie można podzielić tablicy na %d " << size << "równych części!";
        cout << "K-ta wartość zostanie wyliczona przez 1 proces.";
    }
}

y = wartosc * my_rank;
z = wartosc * my_rank + wartosc;
for (i = y; i < z; i++) {
    tab[i] = i + 1;
}

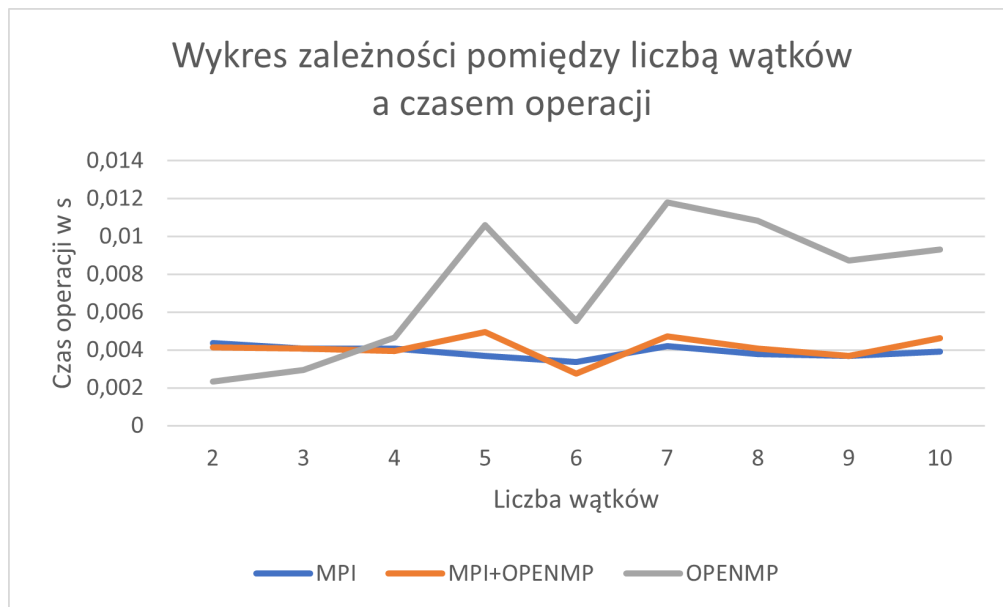
#pragma omp parallel for num_threads(size)
for (int i = y; i < z; i++)
    nums.push_back(rand() % 100000);

```

5 Cząsy operacji

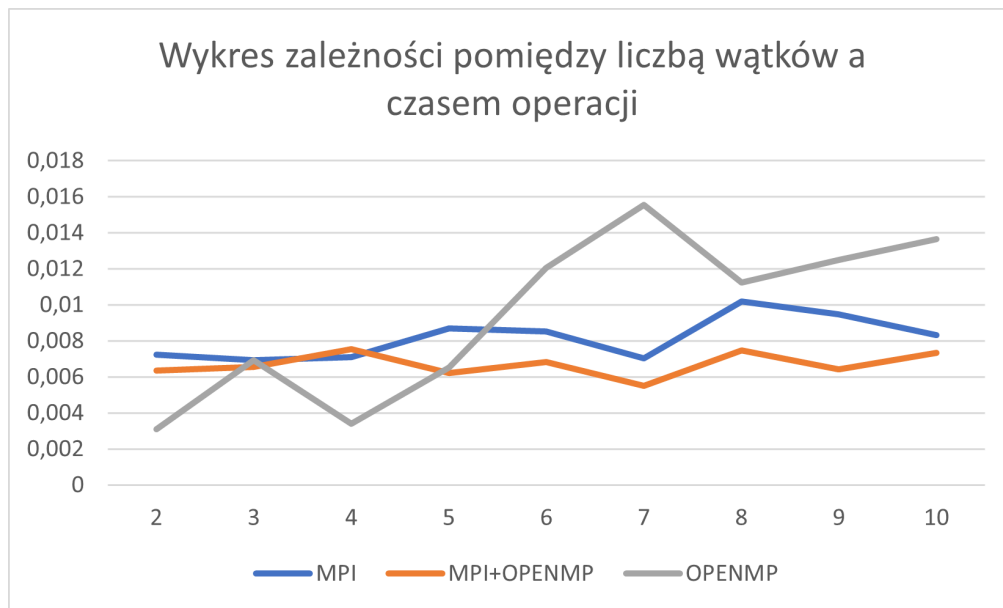
Cząsy wykonania operacji zostały sprawdzone dla 5 tysięcy elementów w tablicy, 10 tysięcy, 20 tysięcy oraz 5 tysięcy elementów tablicy.

5.1 N=5000



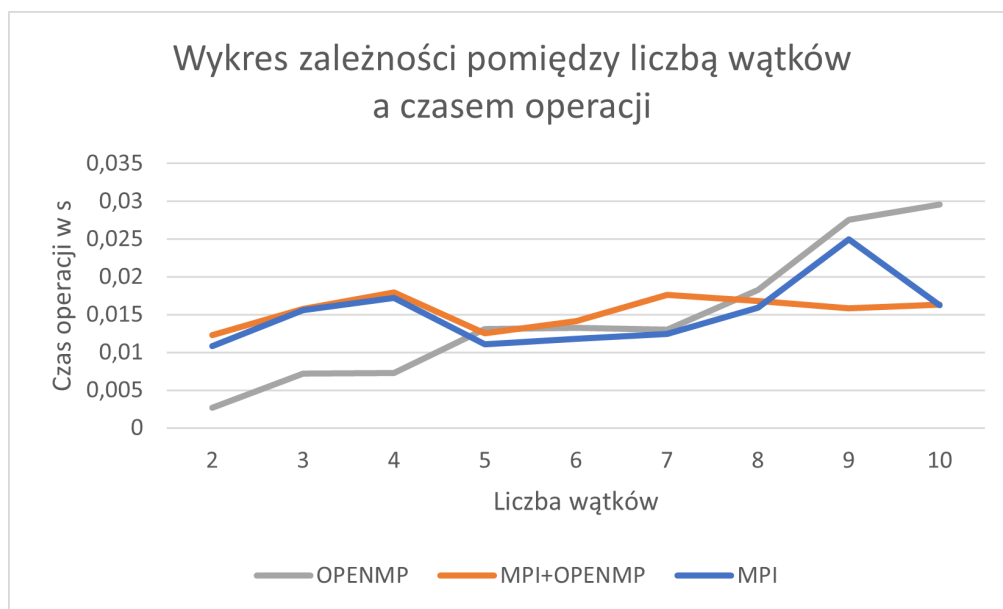
Dla 5 tysięcy elementów w tablicy optymalne są algorytm MPI oraz MPI+OPENMP. Analizując sprawności algorytmów najwyższą wartość osiągnął model OPENMP dla 7 wątków.

5.2 N=10000



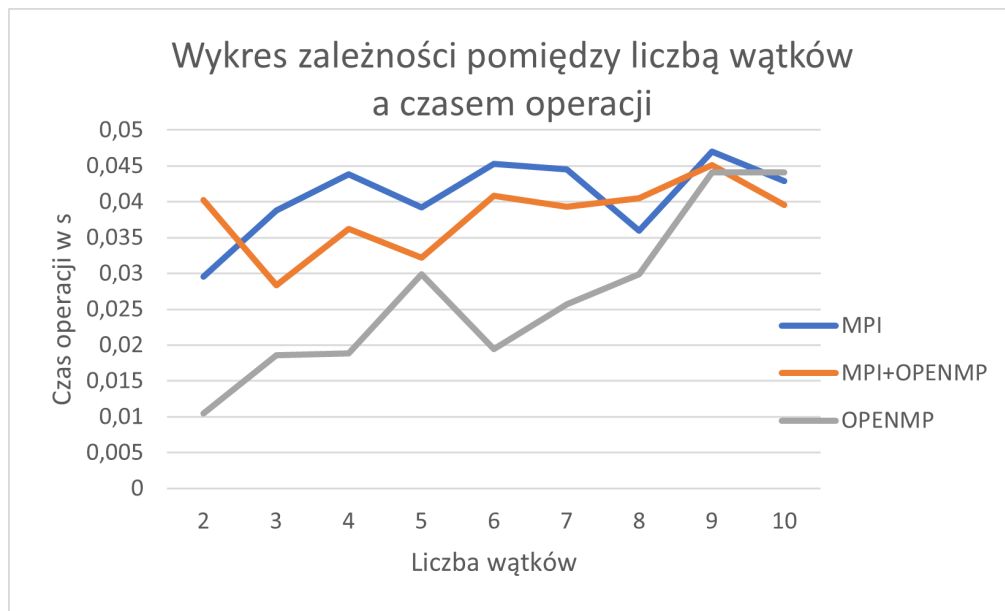
Dla 10 tysięcy elementów w tablicy optymalny jest algorytm MPI+OPENMP. Analizując sprawności algorytmów najwyższą wartość osiągnął model OPENMP dla 7 wątków.

5.3 N=20000



Przy 20 tysiącach elementów w tablicy jest trudność w rozpoznaniu optymalnego modelu programowania równoległego. Analizując sprawności algorytmów najwyższą wartość osiągnął model OPENMP dla 10 wątków.

5.4 N=50000



Przy 50 tysięcy elementów w tablicy optymalny jest algorytm OPENMP. Analizując sprawności algorytmów najwyższą wartość osiągnął model MPI dla 9 wątków.

6 Podsumowanie

Dla testowanych wartości N trudno było zidentyfikować najszybszy algorytm. Jednak należy wiedzieć iż najniższy czas wykonania algorytmu nie wiąże się z najwyższą sprawnością modelu.