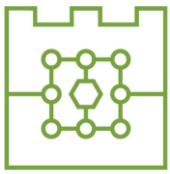


Wydział Informatyki
i Telekomunikacji

PROJEKT Z PRZEDMIOTU PRIR

POLITECHNIKA KRAKOWSKA

Grupa projektowa:

Paweł Malec,

Dominik Kulikowski

Spis treści

1 Wprowadzenie **1**
1.1 Treść projektu zaliczeniowego 1
1.2 Dane wejściowe (tablice) 1
1.3 Specyfikacja maszyny obliczeniowej 2

2 Obliczenia sekwencyjne **3**
2.1 Wprowadzenie teoretyczne 3

3 Obliczenia w MPI **3**
3.1 Wprowadzenie teoretyczne 3

4 Obliczenia w OpenMP **4**
4.1 Wprowadzenie teoretyczne 4

5 Czas wykonania programu **5**
6.1 Dla 2000000 (2mln) 6
6.2 Dla 1000000 (1mln) 7
6.3 Dla 250000 (250tys) 8

1 Wprowadzenie

Celem projektu jest przedstawienie różnic w czasie obliczania średniej aryt- metycznej po uprzednim znalezieniu i wykluczeniu wartości min, max pewnego zbioru liczb (10 elementowego) w różnych modelach programowania równoległego. Na samym początku został przedstawiony program wykonujący obliczenia sekwencyjnie. W dalszej części projektu został przedstawiony model openMP oraz MPI w języku C++. Programy były testowane zaś na uczelnianym serwerze Torus.

1.1 Treść projektu zaliczeniowego

Zbieramy pomiary w 10 elementowe tablice. Następnie znajdujemy wartości min oraz max i je wykluczamy. Pomiarów powinno być minimum 10tys. Na koniec obliczana jest średnia 8 elementowego zbioru (2 wartości zostaną wykluczone). Pomiary sprawdzić przy różnej ilości tablic (min. 10tys) oraz z różną ilością wątków (w przypadku zrównoleglenia).

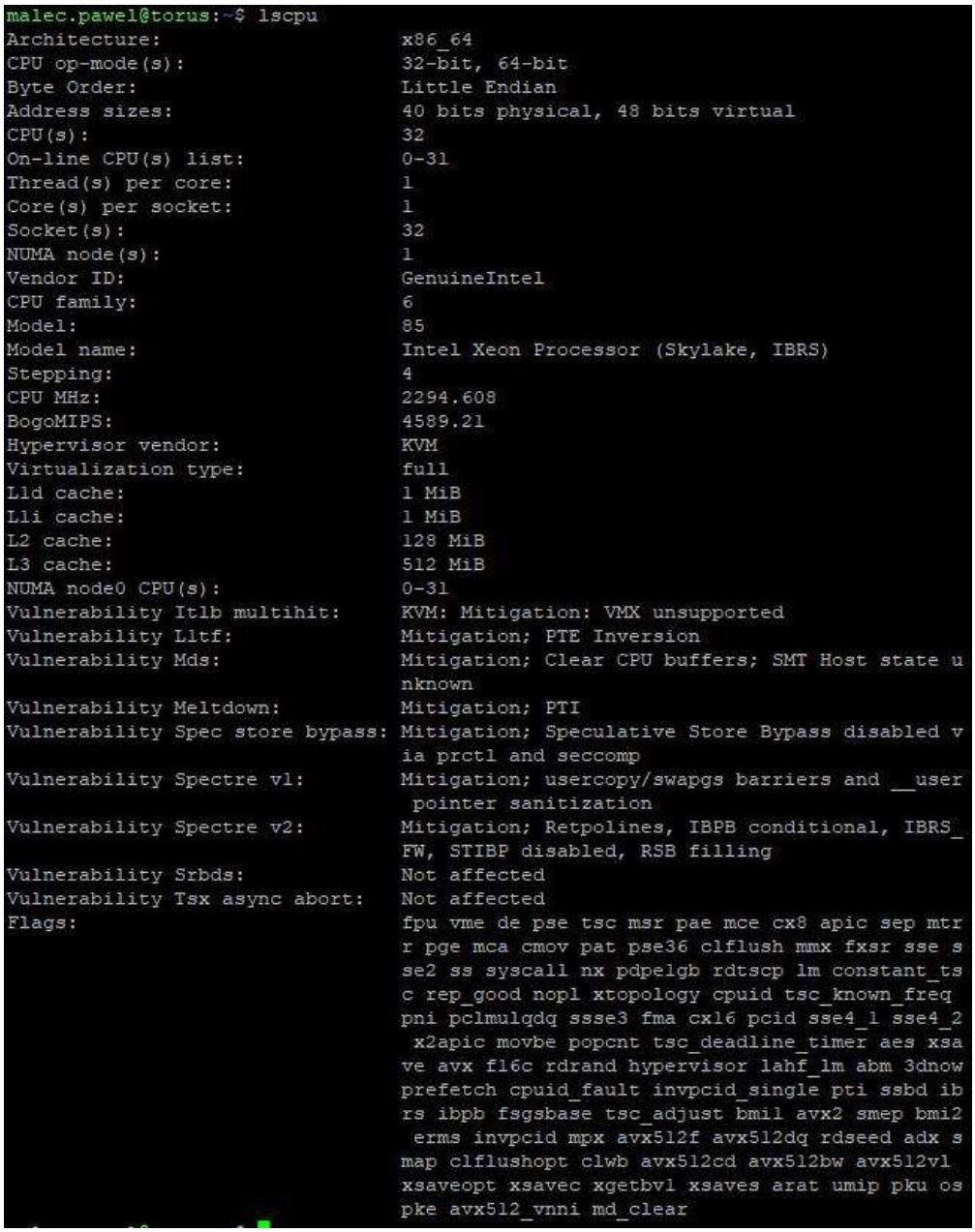
1.2 Dane wejściowe (tablice)

Każda z wersji algorytmu posiada jednakowe dane wejściowe. Są one losowa- ne generatorem pseudolosowych liczb. Poniżej przykład funkcji generującej liczby w wersji sekwencyjnej:

```
int* createTable() {
    int* table = (int*)malloc(sizeof(int) * N);
    srand((unsigned) time(NULL));
      for (int i=0; i < N; i++) {
          table[i] = rand();
    }
    return table;
}
```

1.3 Specyfikacja maszyny obliczeniowej

Wszystkie implementacje algorytmu zostały uruchomione i przetestowane na serwerze Torus, który posiada następującą specyfikację (sprawdzoną za pomocą komendy lscpu):



```
malec.pawel@torus:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   40 bits physical, 48 bits virtual
CPU(s):                          32
On-line CPU(s) list:             0-31
Thread(s) per core:              1
Core(s) per socket:              1
Socket(s):                       32
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           85
Model name:                      Intel Xeon Processor (Skylake, IBRS)
Stepping:                        4
CPU MHz:                         2294.608
BogoMIPS:                        4589.21
Hypervisor vendor:               KVM
Virtualization type:             full
L1d cache:                       1 MiB
L1i cache:                       1 MiB
L2 cache:                        128 MiB
L3 cache:                        512 MiB
NUMA node0 CPU(s):               0-31
Vulnerability Itlb multihit:     KVM: Mitigation: VMX unsupported
Vulnerability L1tf:              Mitigation; PTE Inversion
Vulnerability Mds:               Mitigation; Clear CPU buffers; SMT Host state u
                                 nknown
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled v
                                 ia prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user
                                  pointer sanitization
Vulnerability Spectre v2:        Mitigation; Retpolines, IBPB conditional, IBRS_
                                 FW, STIBP disabled, RSB filling
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Not affected
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep mtr
                                 r pge mca cmov pat pse36 clflush mmx fxsr sse s
                                 se2 ss syscall nx pdpelgb rdtscp lm constant_ts
                                 c rep_good nopl xtopology cpuid tsc_known_freq
                                 pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2
                                  x2apic movbe popcnt tsc_deadline_timer aes xsa
                                 ve avx f16c rdrand hypervisor lahf_lm abm 3dnow
                                 prefetch cpuid_fault invpcid_single pti ssbd ib
                                 rs ibpb fsgsbase tsc_adjust bmi1 avx2 smep bmi2
                                  erms invpcid mpx avx512f avx512dq rdseed adx s
                                 map clflushopt clwb avx512cd avx512bw avx512vl
                                 xsaveopt xsavec xgetbv1 xsaves arat umip pku os
                                 pke avx512_vnni md_clear
```

2 Obliczenia sekwencyjne

2.1 Wprowadzenie teoretyczne

Wersja sekwencyjna programu charakteryzuje się tym, że jest uruchamianana jednym procesorze maszyny czyli standardowo jak każdy kod kompilowany w C++ za pomocą kompilatora gcc. Kolejne instrukcje programu wykonują się kolejno jedna po drugiej, w taki sposób że w jednej jednostce czasu wykonywana jest jedna instrukcja. Nie jest to efektywne w wypadku dużej ilości instrukcji.

```
int sekwencyjny(int* table) {
  int posmax=0, posmin=0, avg=0, sum=0;
    int max = table[0]; // 1 element tablicy
    for (int i = 1; i < N; i++) {
        if(table[i]>max)
            {
            max=table[i];
            posmax=i;
        }
    }
    // usuwanie wartosci max
    for ( int c = posmax; c < N - 1; c++) {
        table[c+1];
    }
    int min = table[0]; // 1 element tablicy
    for (int i = 1; i < N - 1; i++) {
        if(table[i]<min)
        {
            min=table[i];
            posmin=i;
        }
    }
    // usuwanie wartosci min
    for ( int c = posmin; c < N - 2; c++) {
        table[c]=table[c+1];
    }
    // liczenie sumy i sredniej
    for (int i = 0; i < N - 2; i++) {
        sum = sum + table[i];
        avg = sum / N - 2;
    }
    return avg;
}
```

3 Obliczenia w MPI

3.1 Wprowadzenie teoretyczne

Model z wymianą komunikatów (Message Passing Interface) charakteryzu-je się podziałem probelmu na podproblemy, które są opracowywane przez odrębne procesy. Podproblemami są np. wartości min oraz max znalezione w zbiorach danych. Na początku kodu zostały wygenerowane dane wejściowe i zapisane do tablicy. W modelu z wymianą komunikatów dane wymieniane pomiędzy procesami przesyłane są za pomocą komunikatów. Wykorzystano domyślny komunikator, który grupuje wszystkie procesy podczas uruchamiania programu. Liczba w MPI_COMM_WORLD czyli komunikatorze nie zmienia się po jego utworzeniu. Liczba ta jest nazywana rozmiarem komunikatora. Jednocześnie każdy proces wewnątrz komunikatora ma swój unikalny numer, który go identyfikuje. Numer ten jest nazywany rangą procesu. Kawałek kodu w MPI widoczny jest poniżej:

```
// wielkosc tablicy
const int n = 10;
int rank, size;
int  i, j, k;
int avg=0, sum=0;
int values[n] = {0};
// tablica do przechowywania liczb
int maxvalue[2] = {0};
int minvalue[2] = {0};
int globalmaxvalue[2] = {0};
int globalminvalue[2] = {0};

MPI::Status status;
    // inicjalizacja MPI
    MPI::Init(argc, argv);
    size = MPI::COMM_WORLD.Get_size();
    rank = MPI::COMM_WORLD.Get_rank();
```

4 Obliczenia w OpenMP

4.1 Wprowadzenie teoretyczne

Model pamięci wspólnej algorytmu obliczającego średnią liczb został wykonany w standardzie OpenMP (z ang. Open Multi-Processing). Standard ten wykorzystuje pracę na wielu wątkach oraz pamięć współdzieloną. Kod algorytmu został napisany w języku C++. Dzięki temu, że standard OpenMP został uzgodniony przez głównych producentów sprzętu i oprogramowania, charakteryzuje się on przenośnością, skalowalnością i prostotą użycia.

Implementacja algorytmu odróżnia się od MPI brakiem przesyłania komunikatów, ponieważ wykorzystana została pamięć wspólna. Kod jest tożsamy z wersją sekwencyjną i bardzo prosty w użyciu co zachęca do korzystania z openMP, jednakże pętla algorytmu staje się zrównoleglona. Dodatkowo dla łatwości zmiany ilości wątków stworzono zmienną „watki". Zrównoleglony kod algorytmu przedstawia rysunek poniżej.

```
int openmp(int* table) {
    printf("\nLiczę dla openmp: \n");
    int posmax=0, posmin=0, avg=0, sum=0;
    #pragma omp parallel for num_threads(watki)
        for (int i = 1; i < N; i++) {
            int max = table[0]; // 1 element tablicy
            if(table[i]>max)
                {
                max=table[i];
                posmax=i;
            }
        }
        // usuwanie wartosci max
        for ( int c = posmax; c < N - 1; c++) {
            table[c]=table[c+1];
        }
        int min = table[0]; // 1 element tablicy
        for (int i = 1; i < N - 1; i++) {
            if(table[i]<min)
            {
                min=table[i];
                posmin=i;
            }
        }
        // usuwanie wartosci min
        for ( int c = posmin; c < N - 2; c++) {
            table[c]=table[c+1];
        }
```

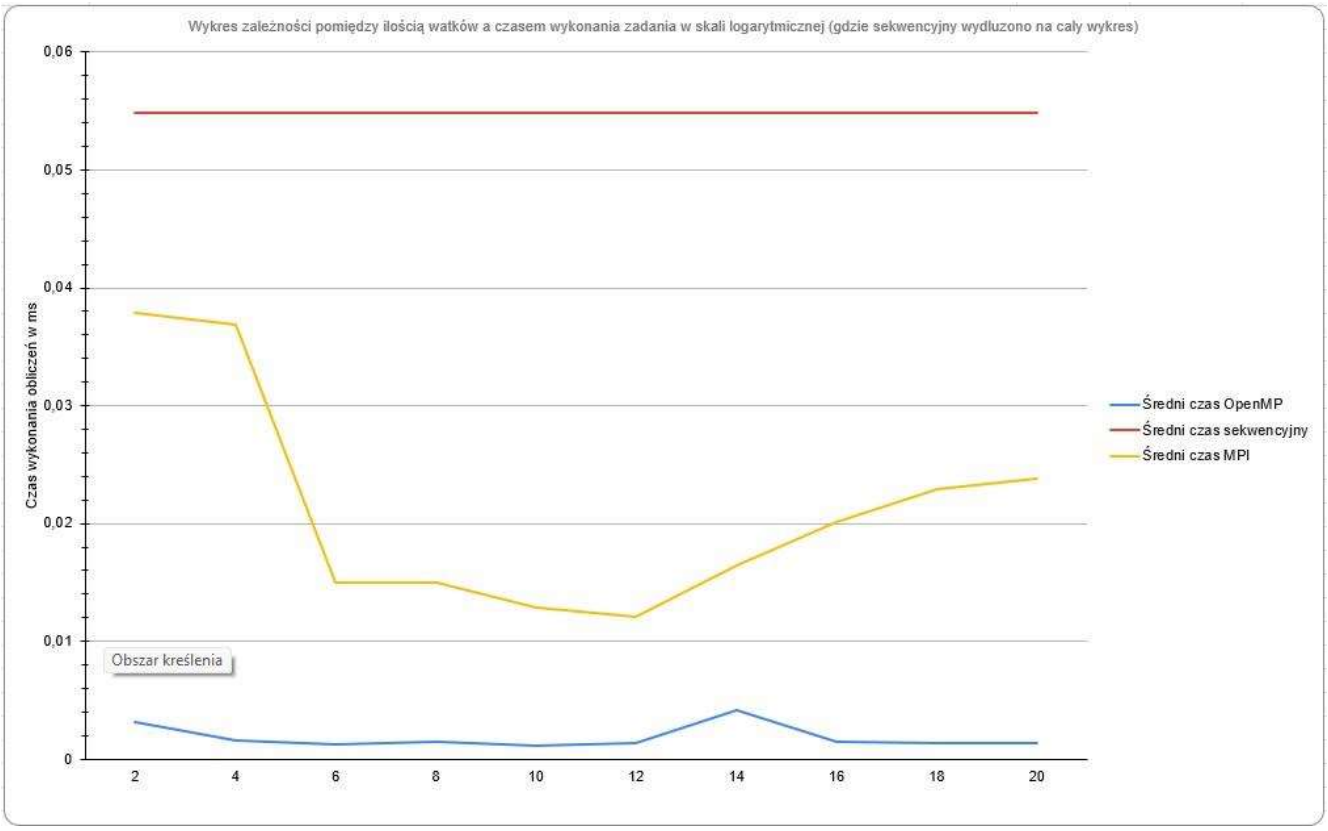
5 Czas wykonania programu

Czasy wykonania algorytmów dla wcześniej opisanych modeli zostały przetestowane dla różnych ilości porcji danych wejściowych: dla 2000000, 1000000 oraz 250000 tablic 10 elementowych. Ilość ich musiała zostać zwiększona z wstępnie założonej 10tys gdyż czasy pomiarów wykonania były zbyt krótkie do analizy.

Wszystkie obliczenia oraz wykresy w sprawozdaniu zostały wykonane w arkuszu kalkulacyjnym MS Excel. Są one także dołączone w katalogu projektu. Należy zaznaczyć, że program sekwencyjny został przedstawiony w ilościach wątku aby na wykresie pojawił się jako linia ciągła (dla każdego z ilości wątków są takie same dane gdyż program sekwencyjny nie posiada wątków).

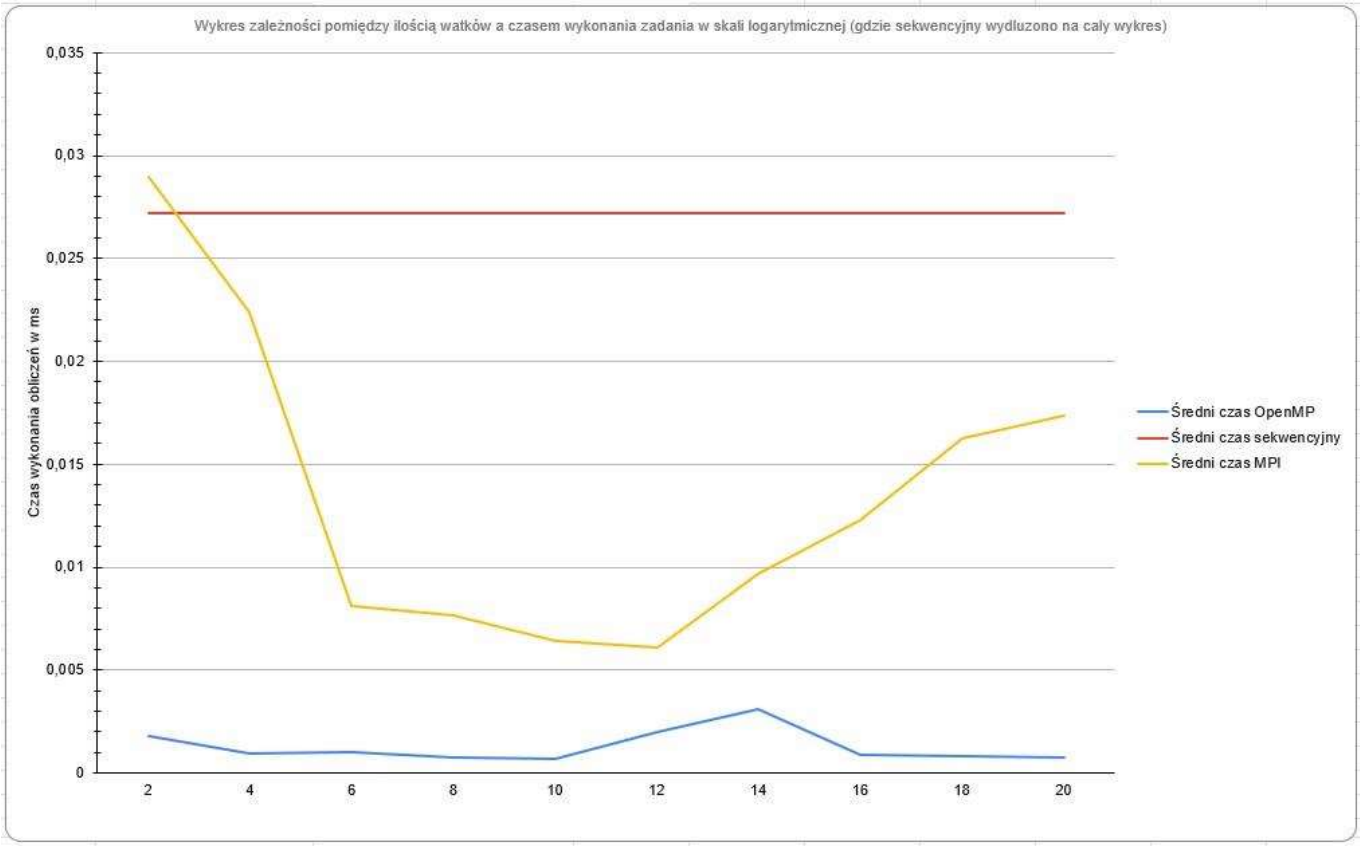
OpenMP						Średnie wyniki w sekundach
Ilość wątków	Próba 1	Próba 2	Próba 3	Próba 4	Próba 5	Średni czas OpenMP
2	0,0029845	0,00303592	0,00310715	0,0031502	0,0032502	0,003105594
4	0,00166938	0,00149998	0,00150127	0,00166708	0,00151465	0,001570472
6	0,00185212	0,00123005	0,00106349	0,00108518	0,00103658	0,001253484
8	0,00149778	0,00144172	0,00147225	0,00144463	0,00144688	0,001460652
10	0,00110988	0,00115699	0,00116054	0,00115914	0,00116454	0,001150218
12	0,00245736	0,00119999	0,00100594	0,000977074	0,000980634	0,0013242
14	0,00171198	0,00167351	0,00335198	0,00730489	0,00653738	0,004115948
16	0,0014658	0,0015184	0,00150844	0,00153335	0,00137871	0,00148094
18	0,00140136	0,00140248	0,0014112	0,00137171	0,00141118	0,001399586
20	0,00131404	0,00136332	0,00126889	0,00128754	0,00133759	0,001314276
sekwencyjny	Próba 1	Próba 2	Próba 3	Próba 4	Próba 5	Średni czas sekwencyjny
2	0,054216	0,053633	0,056212	0,054441	0,056055	0,0549114
4	0,054216	0,053633	0,056212	0,054441	0,056055	0,0549114
6	0,054216	0,053633	0,056212	0,054441	0,056055	0,0549114
8	0,054216	0,053633	0,056212	0,054441	0,056055	0,0549114
10	0,054216	0,053633	0,056212	0,054441	0,056055	0,0549114
12	0,054216	0,053633	0,056212	0,054441	0,056055	0,0549114
14	0,054216	0,053633	0,056212	0,054441	0,056055	0,0549114
16	0,054216	0,053633	0,056212	0,054441	0,056055	0,0549114
18	0,054216	0,053633	0,056212	0,054441	0,056055	0,0549114
20	0,054216	0,053633	0,056212	0,054441	0,056055	0,0549114
MPI	Próba 1	Próba 2	Próba 3	Próba 4	Próba 5	Średni czas MPI
2	0,037234	0,0372354	0,038214	0,0392314	0,03732	0,03784696
4	0,036552	0,036185	0,03652	0,038375	0,036572	0,0368408
6	0,014886	0,015126	0,015118	0,014774	0,014855	0,0149518
8	0,015106	0,014659	0,014586	0,015319	0,015134	0,0149608
10	0,012995	0,012813	0,013043	0,012331	0,013018	0,01284
12	0,012102	0,011893	0,01181	0,012226	0,012267	0,0120596
14	0,018592	0,012101	0,012377	0,014326	0,02481	0,0164412
16	0,020916	0,015426	0,021631	0,020013	0,022829	0,020163
18	0,019719	0,020614	0,030609	0,020885	0,022603	0,022886
20	0,020942	0,023831	0,019211	0,033376	0,021797	0,0238314

6.1 Dla 2000000 (2mln)



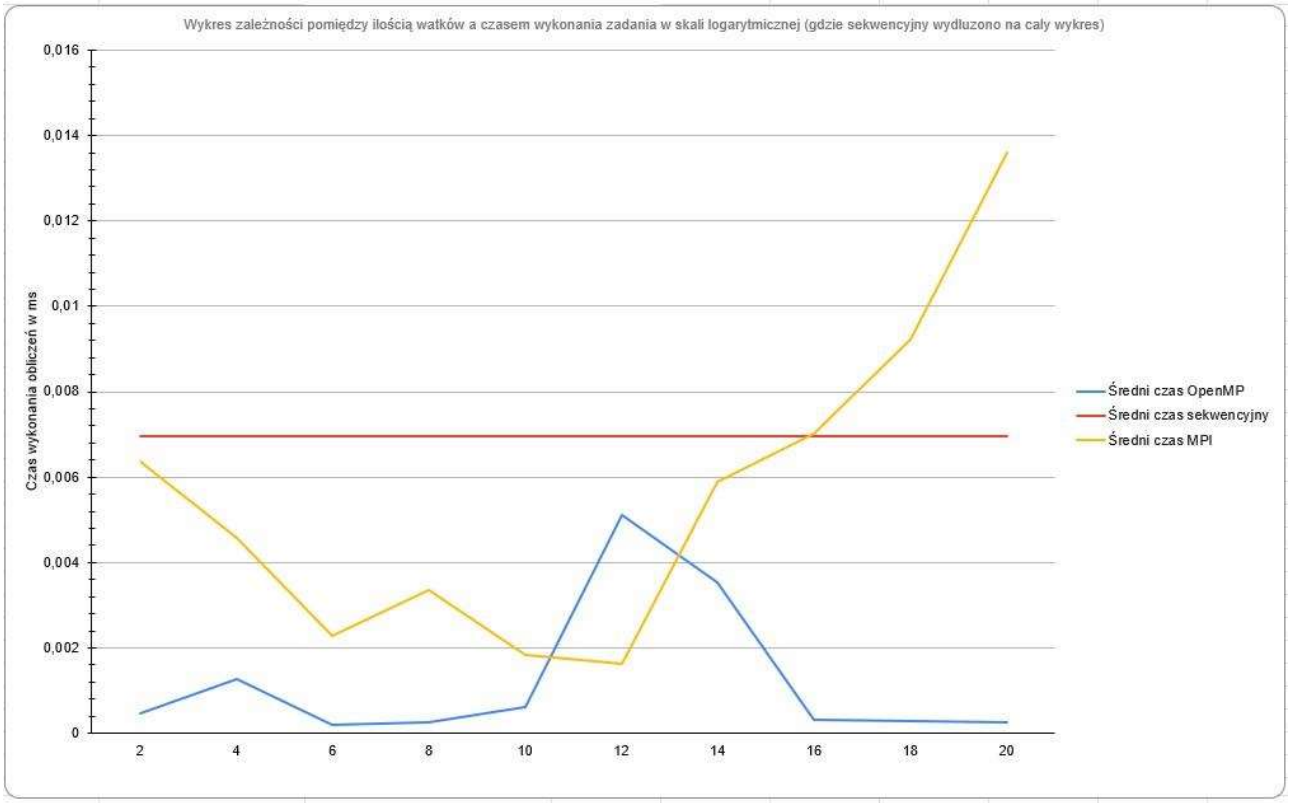
Dla 2 mln tablic optymalny okazał się algorytm wykonany w technologii OpenMP.Algorytm sekwencyjny znacząco oddala się od MPI oraz OpenMP.

6.2 Dla 1000000 (1mln)



Dla 1 mln tablic optymalny okazał się algorytm wykonany w technologii OpenMP. W tym przypadku jednak na samym początku przy małej ilości wątków MPI potrafi być przegoniony przez program sekwencyjny co wskazuje, że nie jest on optymalny przy małych obliczeniach.

6.3 Dla 250000 (250tys)



W ogólnym przypadku dla 250 tys. wartości w tablicy wejściowej optymalnyjest algorytm OpenMP. Jednak dla 12 wątków najszybszy okazał się algorytm MPI. Analizując sprawności algorytmów najwyższą wartość osiągnął model OpenMP dla ilości 6 wątków. Ponownie przy małej ilości wątków czasy zrównują się z wykonaniem sekwencyj