Tomasz Ligęza

Programowanie równoległe. Przetwarzanie równoległe i rozproszone.

Sprawozdanie z laboratorium 13.

Cel zajęć:

> Doskonalenie umiejętności analizy wydajności programów równoległych.

W ramach zajęć zrealizowałem następujące kroki:

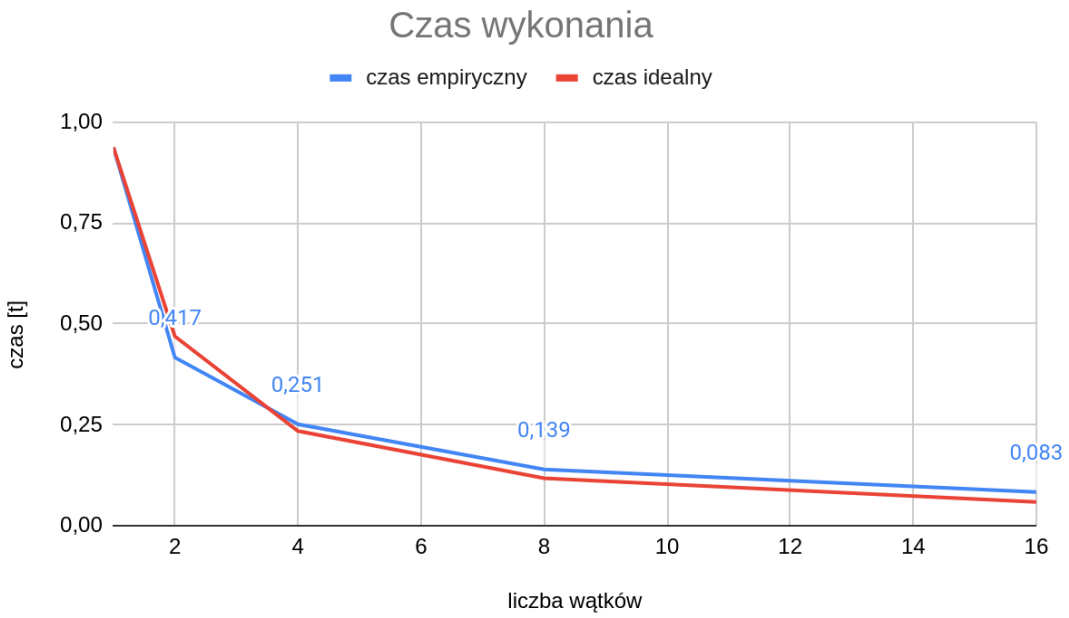
1. Utworzyłem katalog roboczy lab_13 i pobrałem pliki źródłowe programów calka_omp.c oraz wchodzących w skład paczki mat_vec_row_MPI_OpenMP.tgz.
2. Przeprowadziłem serię eksperymentów wydajnościowych programu do obliczania całki.

Skompilowałem kod i w celu zautomatyzowania tego procesu napisałem krótki skrypt, który następnie uruchomiłem trzykrotnie.

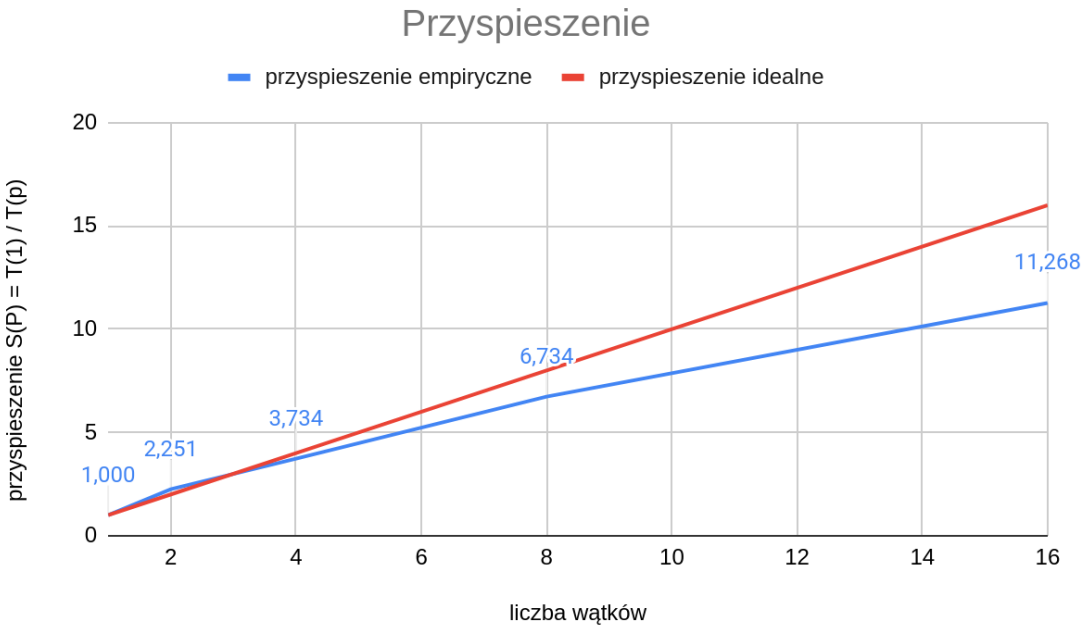
```
#!/bin/bash

export OMP_NUM_THREADS=1

./a.out > wynik.txt

export OMP_NUM_THREADS=2

./a.out >> wynik.txt

export OMP_NUM_THREADS=4

./a.out >> wynik.txt

export OMP_NUM_THREADS=8

./a.out >> wynik.txt

export OMP_NUM_THREADS=16

./a.out >> wynik.txt
```

Następnie wyniki przeniosłem do arkusza kalkulacyjnego, w którym wykonałem wykresy liniowe dla uzyskanych wyników. Wyniki zamieszczam w tabeli. Prezentują się one następująco:

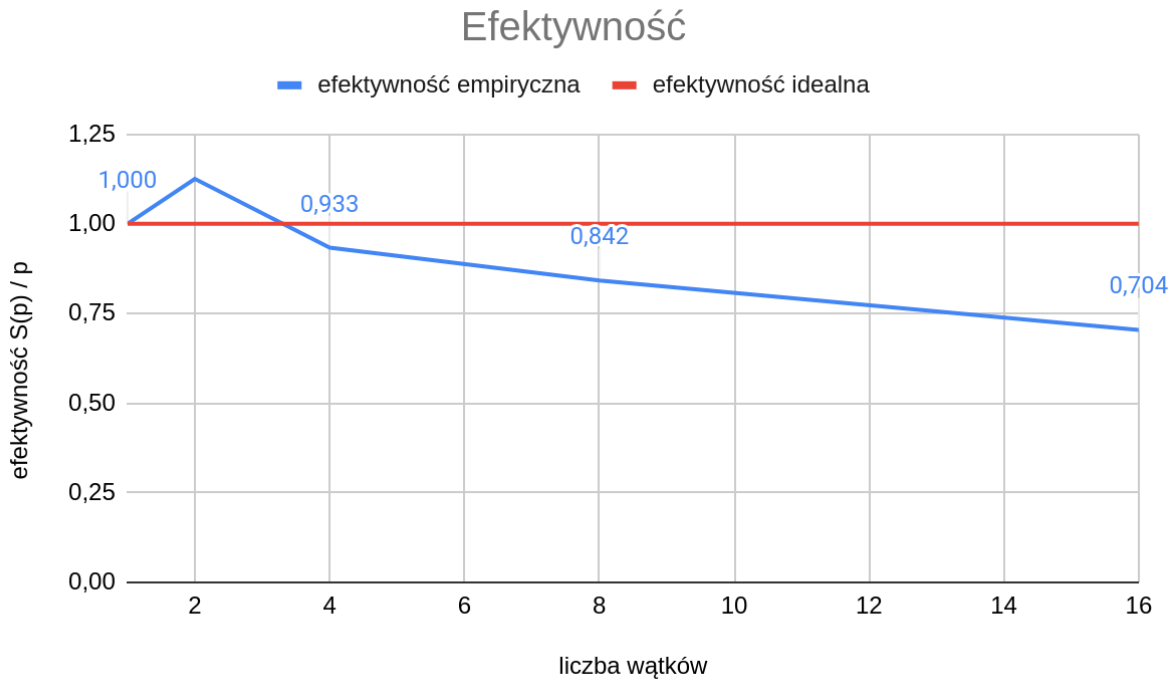
liczba wątków	czas empiryczny [s]	czas idealny [s]
1	0,937683	0,937683
2	0,41659	0,4688415
4	0,251138	0,23442075
8	0,139242	0,117210375
16	0,083215	0,0586051875



Czas wykonania zmierzony podczas eksperymentu niemal zlewa się z idealnym określonym na podstawie pierwszego wyniku. Czas wykonania z użyciem dwóch wątków jest niższy niż ten idealny - widocznie maszyna wykonująca program z użyciem jednego wątku była obciążona jeszcze innymi zadaniami, gdyż niemożliwe jest, żeby wydajność wzrosła więcej niż dwukrotnie wraz z dwukrotnym zwiększeniem liczby wątków. Świadczy to o błędzie pomiarowym.



Na powyższym wykresie widzimy jak nasze przyspieszenie wzrasta wraz ze zwiększeniem ilości wątków wykonujących program. Sytuacją idealną byłaby taka, gdzie nasze przyspieszenie wynosi tyle ile razy zwiększyliśmy ilość wątków biorących udział w obliczeniach.

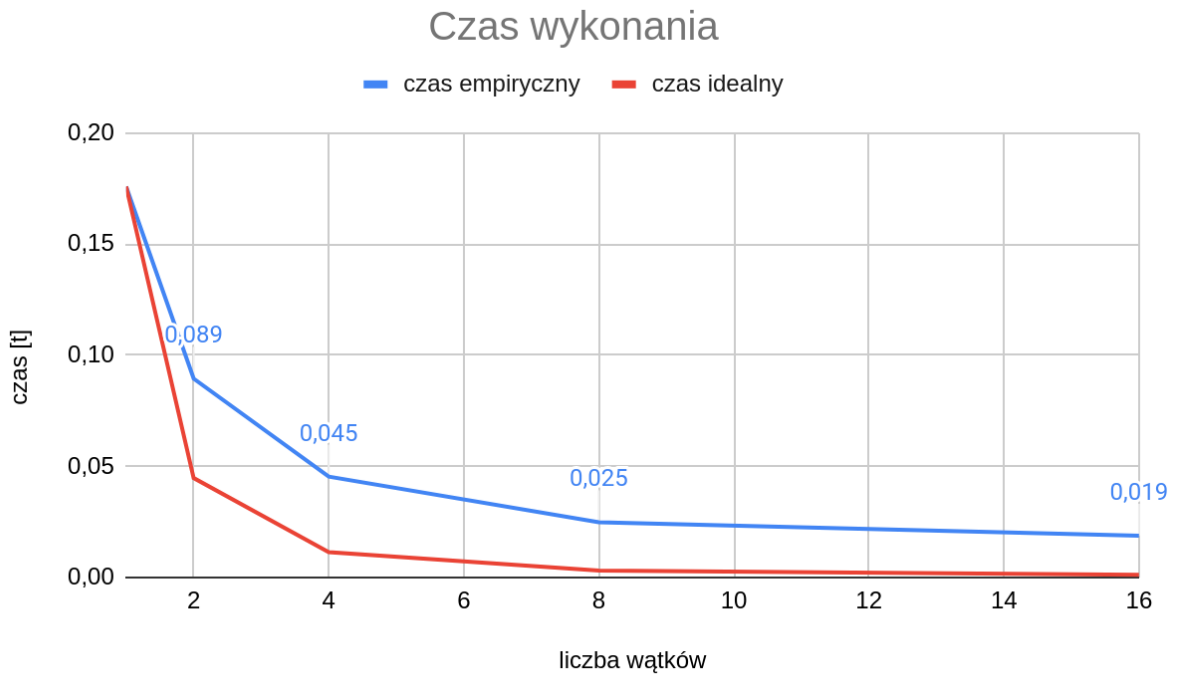


Bardzo dobrze widać tutaj błąd pomiaru przy użyciu dwóch wątków, gdzie efektywność wynosi więcej niż maksymalna wielkość 1.0.

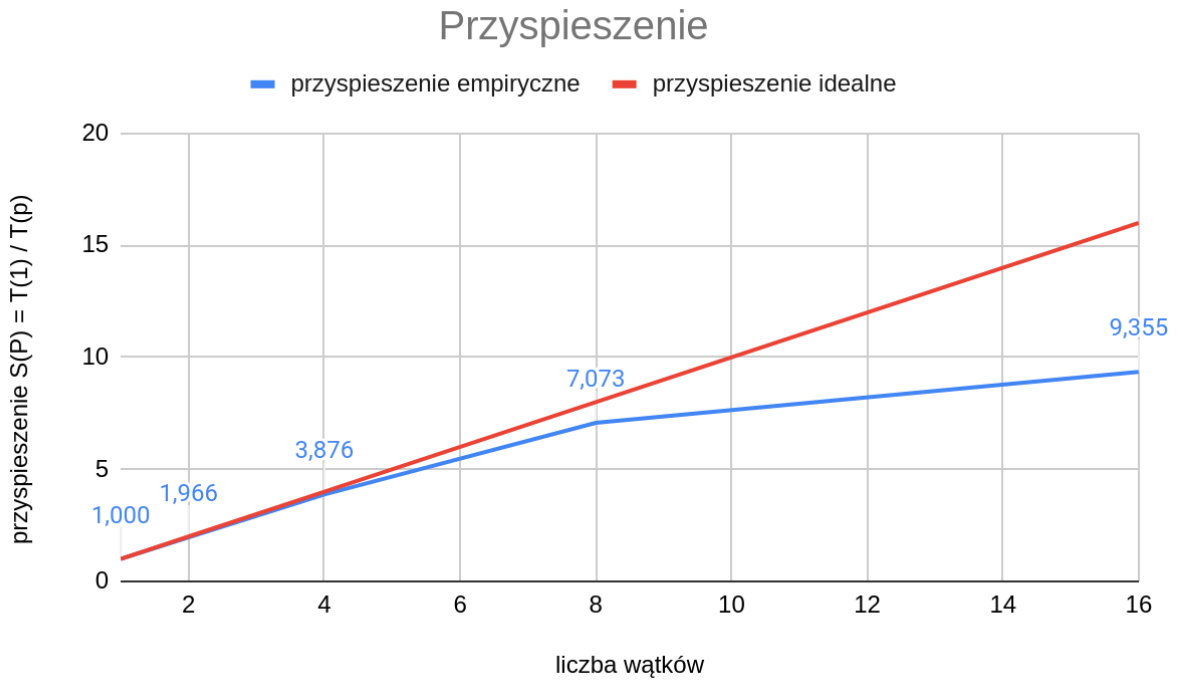
3. Następnie przeprowadziłem serię eksperymentów wydajnościowych programu do obliczania iloczynu macierz-wektor w środowisku MPI (dla wymiaru 12144). Tak prezentują się wyniki:

liczba wątków	czas empiryczny [s]	czas idealny [s]
1	0,175895	0,175895
2	0,089452	0,044726
4	0,045378	0,0113445
8	0,024868	0,0031085
16	0,018802	0,001175125

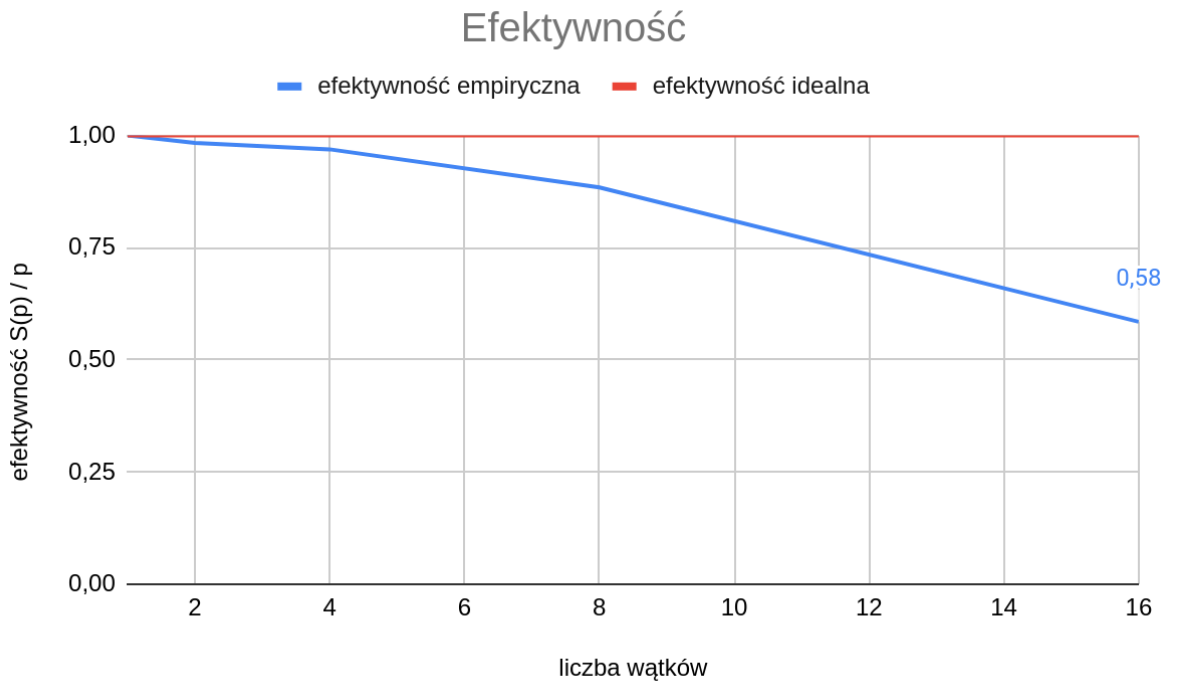
Kolejnym krokiem było przygotowanie wykresów prezentujących wyniki:



W tym przypadku wyniki nie są już takie dobre jak w przypadku programu obliczającego całkę. Czasy wykonania mierzone eksperymentalnie znacząco przewyższają wartości idealne.



Najniższy przyrost przybliżenia dostrzec można przy 16 wątkach. Uzyskaliśmy przyspieszenie o przybliżonej wartości 2, co w warunkach idealnych oznaczałoby, że “dołożyliśmy” tylko pracę dwóch dodatkowych wątków.



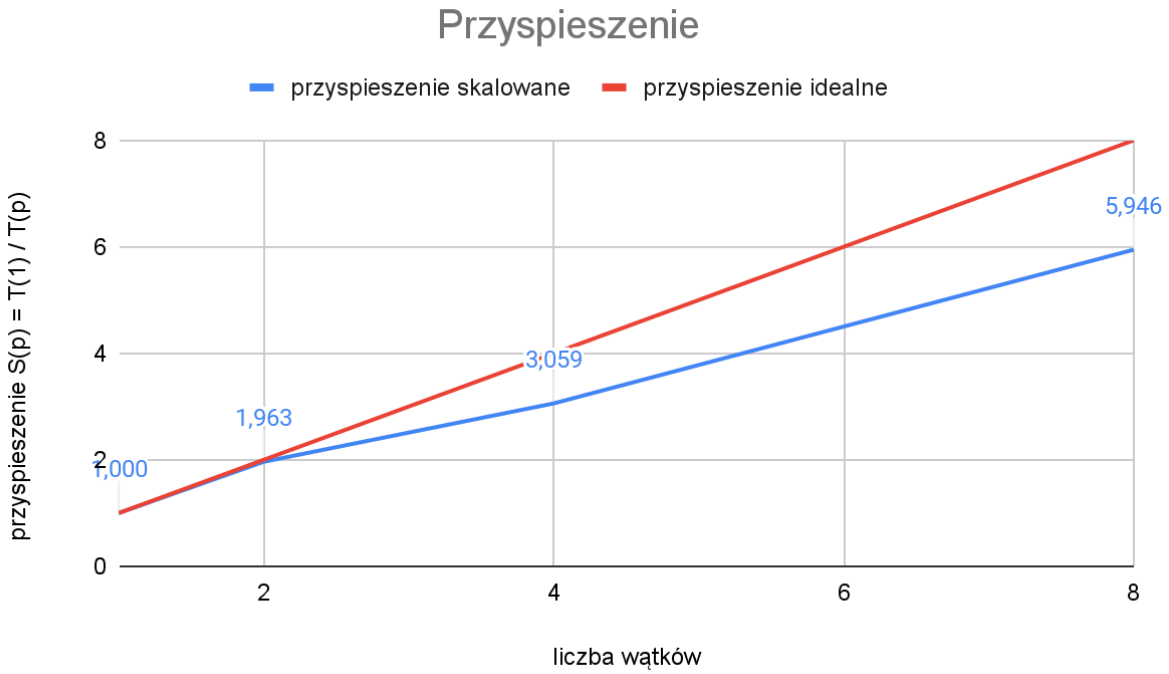
Efektywność w ostatnim przypadku (16 wątków) zdecydowanie spadła. Wynosi ona w tym przypadku tylko 58% wartości idealnej.

4. Następnym krokiem było przeprowadzenie testu skalowalności w sensie słabym - każdy wątek miał taki sam rozmiar zadania.

Poniżej prezentuję wyniki dla poszczególnych wykonań: (pomiary zrealizowałem tylko dla uruchomienia sekwencyjnego oraz w przypadkach, gdy mnożnik ilości operacji równy był ilości wątków).

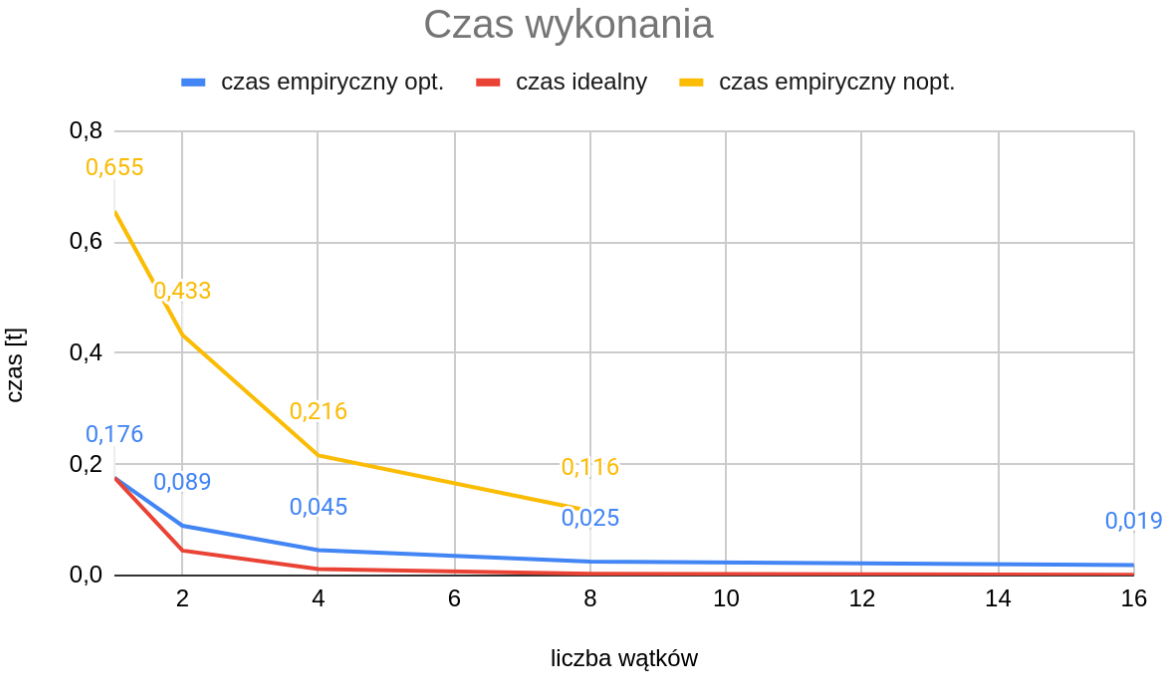
	ilość operacji zmiennoprzecinkowych (W)			
ilość wątków	W	2W	4W	8W
1	0,013508	0,026978	0,042963	0,085427
2	-	0,013742	-	-
4	-	-	0,014044	-
8	-	-	-	0,014368

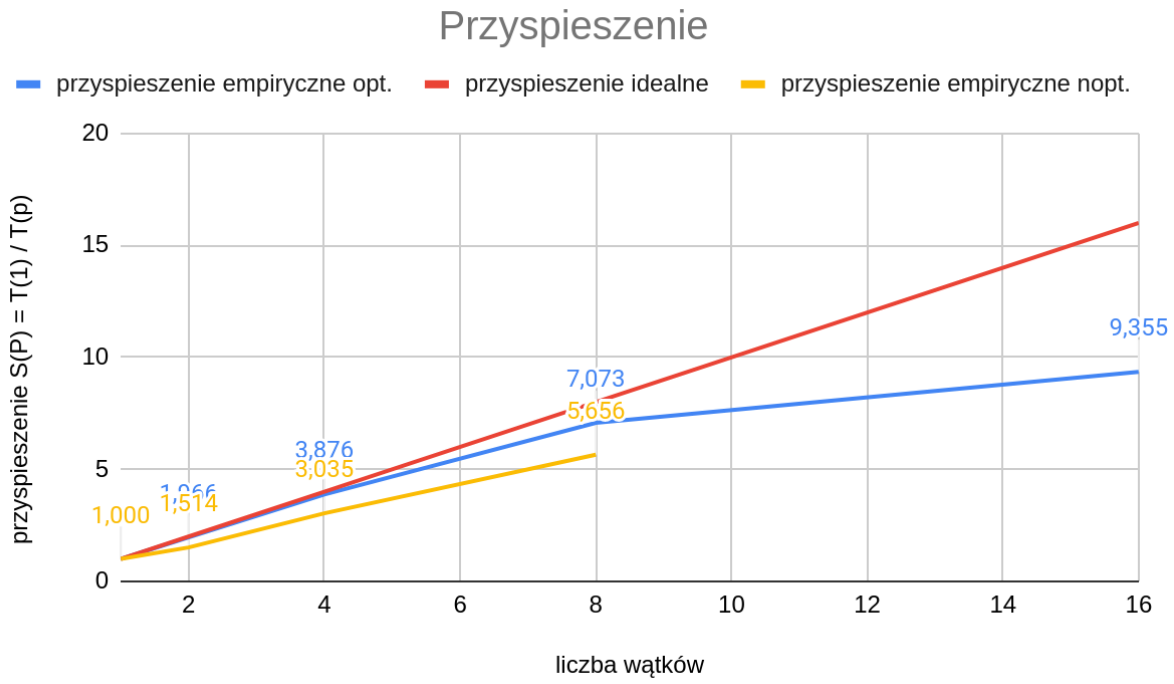
Na powyższym wykresie prezentuję czas wykonania obliczeń o rozmiarze p * W_0 przez p wątków.

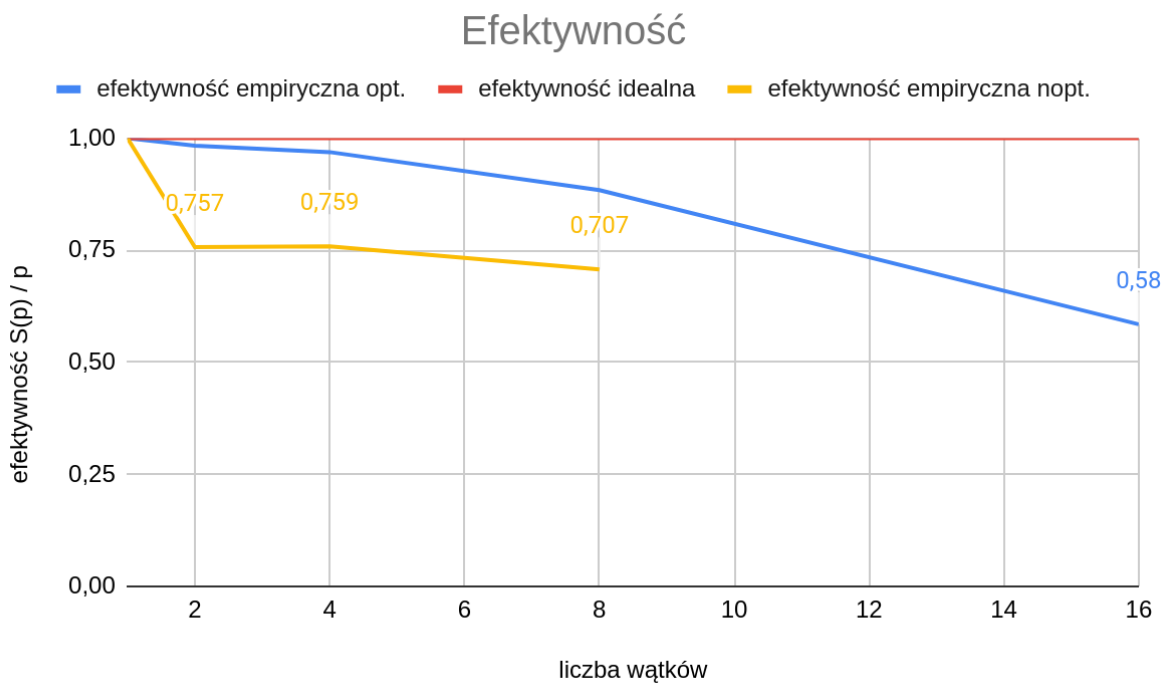


Powyższy wykres prezentuje przyspieszenie skalowane, wywodzące się ze wzoru:
$Ss(p) = T(p*W_0, 1) / T(p*W_0, p)$.

Ostatnim realizowanym przeze mnie eksperymentem było zbadanie skalowalności w sensie silnym, ale dla programu nie zoptymalizowanego. Poniżej zaprezentuję porównane wyniki programu zoptymalizowanego oraz nie zoptymalizowanego na jednym wykresie:







Wnioski:

Badanie wydajności programów równoległych kodu w sposób genialny pokazują jak dużo daje nam programowanie współbieżne. Nauczyło mnie również jak przeanalizować wpływ zrównoleglenia na czas wykonywania programu oraz jak w dobry sposób to zobrazować.

Ostatnie zaprezentowane przeze mnie wyniki pokazały jak optymalizacja kodu przez kompilator daje duże rezultaty - czas wykonania kodu zoptymalizowanego zmniejszył się ponad trzykrotnie. Niestety wyniki nie pokazały tego, że kod nie zoptymalizowany skaluje się lepiej niż ten “ulepszony” przez kompilator.