Grupa lab.

5

Data wykonania:

08.01.2024

Temat: Wydajność

Imie i nazwisko:

Łukasz Sawina

Cel:

Celem ćwiczenia było poznanie sposobów analizy wydajności programów równoległych.

Program obliczający całkę

Pierwszy program, który dostaliśmy wykonuje obliczanie całki. Program napisany jest z wykorzystaniem programowania równoległego openMP.
Zadaniem naszym było porównać czas wykonania w zależności od ilości wątków, które wykonują dany program oraz przyspieszenie.
Do pomiarów zostały użyte 1, 2 oraz 4 wątki, ponieważ pomiary zostały wykonane na maszynie posiadającej procesor 4 rdzeniowy/8 wątkowy. Program został również skompilowany z flagą -O0 w celu uniknięcia optymalizacji.

Wyniki pomiarów

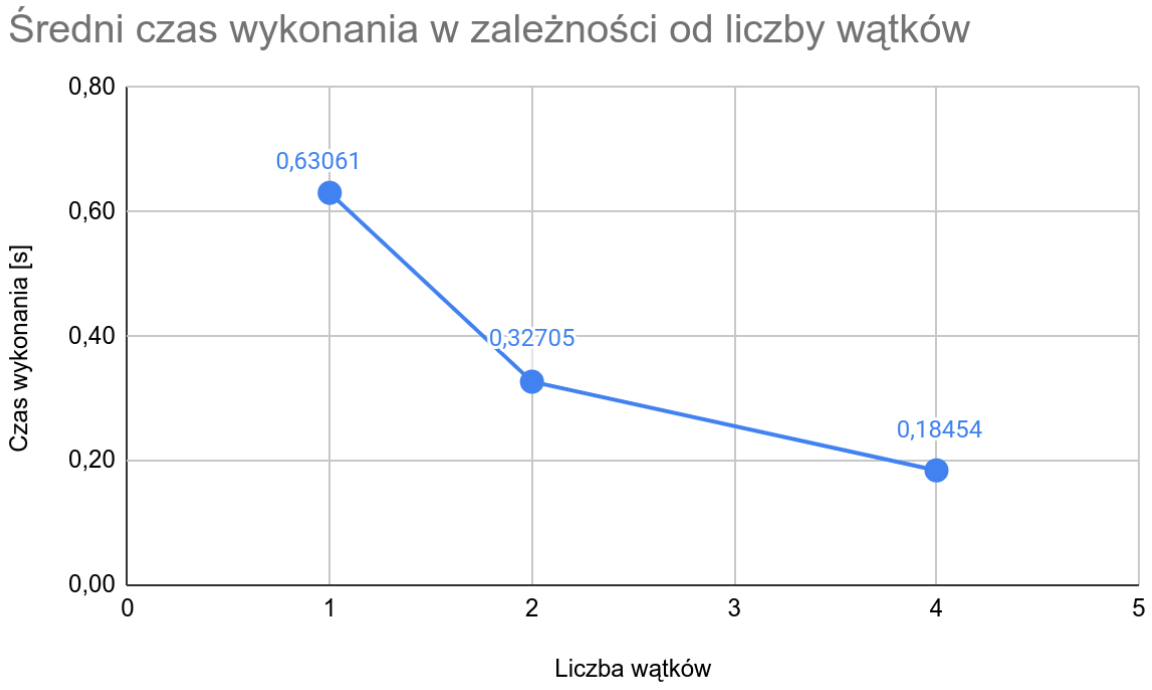
Liczba wątków	Czas wykonania [s]
1	0,64530
1	0,62653
1	0,61464
1	0,58900
1	0,63596
2	0,30769
2	0,34270
2	0,32081
2	0,32727
2	0,33680

4	0,18467
4	0,17667
4	0,18403
4	0,19278
4	0,15853

Po odrzuceniu wartości, które odstają od reszty wyników (zaznaczone na czerwono) obliczamy średni czas wykonania dla każdej wersji programu.

Średnie	
Liczba wątków	Czas wykonania [s]
1	0,63061
2	0,32705
4	0,18454

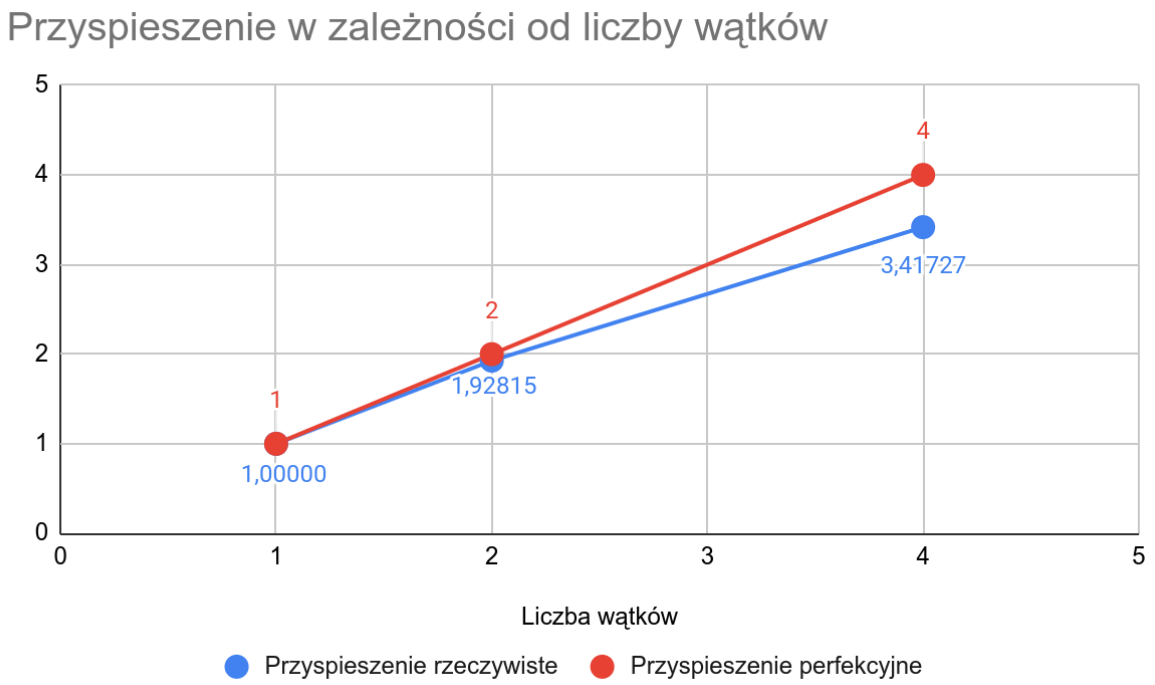
Jak można zauważyć czas wykonania drastycznie maleje wraz ze wzrostem ilości wątków, udało się prawie osiągnąć tylu krotne przyspieszenie ile wątków zostało użytych. Dobrze obrazuje to poniższy wykres:



Następnie badamy przyspieszenie oraz efektywność naszego programu w zależności od ilości wątków.
Do obliczenia przyspieszenia porównujemy stosunek średniego czasu wykonania dla 1 wątku do czasu wykonania dla w wątków (przyspieszenie względne).

Liczba wątków	Przyspieszenie rzeczywiste	Przyspieszenie perfekcyjne
1	1,00000	1
2	1,92815	2
4	3,41727	4

Dla dodatkowego porównania w trzeciej kolumnie zawarte jest perfekcyjne przyspieszenie, które chcielibyśmy osiągnąć po zrównolegleniu. Zauważyć można, że otrzymane przez nas przyspieszenie jest bardzo zbliżone do perfekcyjnego. Podobieństwo dobrze obrazuje poniższy wykres.



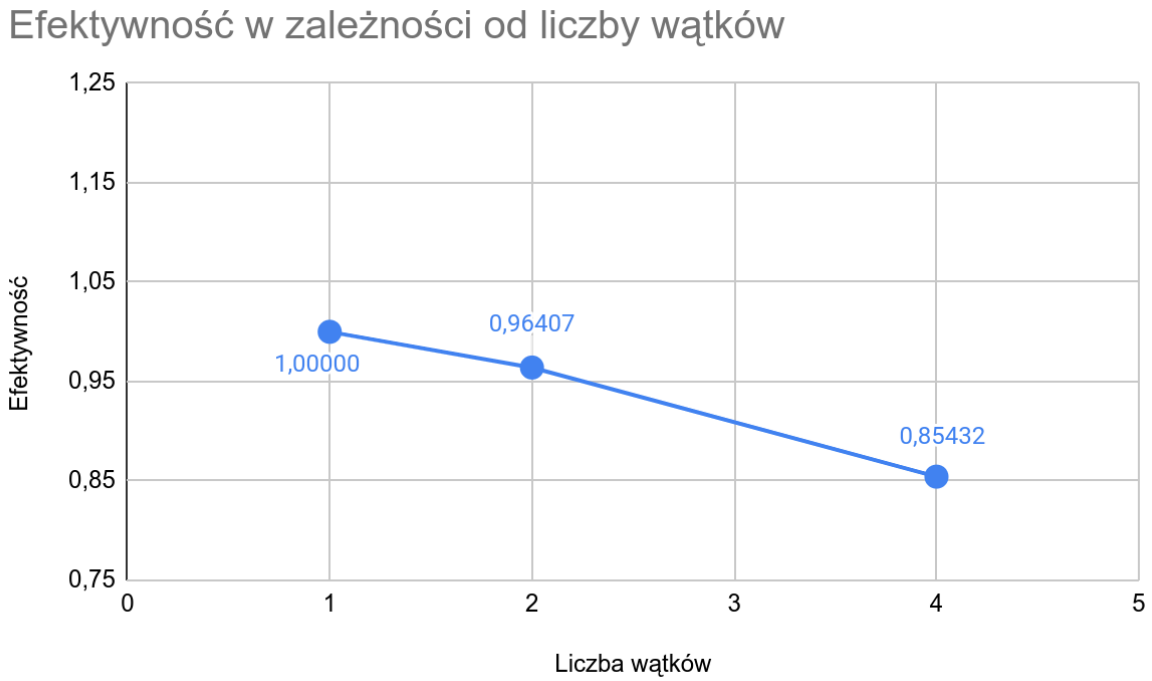
Oczywistym faktem jest, że nie uda się nam uzyskać idealnego przyspieszenie, głównie przez fakt dodatkowych narzutów przy tworzeniu oraz zarządzaniu wątkami. Przy wykonaniu programu wielowątkowo, każdy wątek musi zostać utworzony, co przede wszystkim powoduje tą różnicę.

Zauważyć tutaj można, że otrzymaliśmy dobrą skalowalność silną, czyli wraz ze wzrostem ilości wątków maleje nam czas wykonania.

W przypadku obliczania efektywności musimy wskazać stosunek ilości wątków do przyspieszenia.

Liczba wątków	Efektywność
1	1,00000
2	0,96407
4	0,85432

Widać pewien spadek efektywności wraz ze wzrostem ilości wątków, ponownie jest to spowodowane narzutami systemu oraz tworzeniem nowych wątków.



Podsumowując dla tego programu liczącego całkę udało się osiągnąć lepszą wydajność programu dla większej ilości wątków, w sytuacji gdyby maszyna miałą większą ilość rdzeni/wątków moglibyśmy śmiało użyć większej ilości wątków do zrównoleglenia, ponieważ można zauważyć niewielki spadek efektywności programu.

Program Mat-Vec

Kolejny program liczący iloczyn macierzy i wektora został napisany z wykorzystaniem MPI. Ponownie jak wcześniej chcemy zbadać jego wydajność w wersji równoległej.

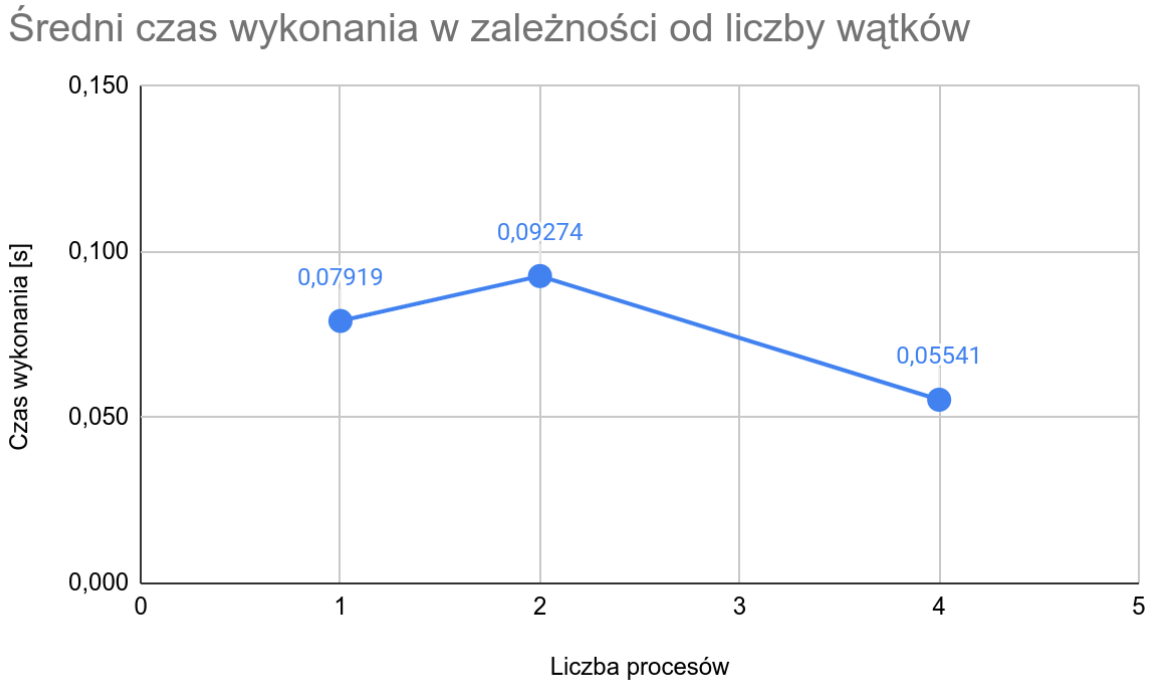
Wyniki pomiarów

Liczba procesów	Czas wykonania [s]
1	0,07872
1	0,07949
1	0,07981
1	0,07897
1	0,07894
2	0,09056
2	0,09389
2	0,09140
2	0,09416
2	0,09367
4	0,05261
4	0,05448
4	0,05599
4	0,05589
4	0,05811

W przypadku tego programu udało się osiągać czasy wykonania w miarę podobne do siebie, dlatego nie musimy żadnego z nich odrzucać.
Ponownie wyliczamy średni czas wykonania każdego z wariantów:

Średnie	
Liczba procesów	Czas wykonania [s]
1	0,07919
2	0,09274
4	0,05541

W tym przypadku można zauważyć skok czasu wykonania dla dwóch procesów, ponieważ tutaj wykorzystujemy inny mechanizm zrównoleglania niż poprzednio, w tym przypadku jednocześnie uruchamiamy kilka procesów, które działają współbieżnie. Prawdopodobnie nagły wzrost czasu wykonania jest spowodowany nieopłacalną ilością procesów do wykorzystywania komunikacji grupowej. Lepszy czas widać w przypadku 4 procesów, tam już wykorzystanie komunikacji grupowej przyspieszyło nasz program.



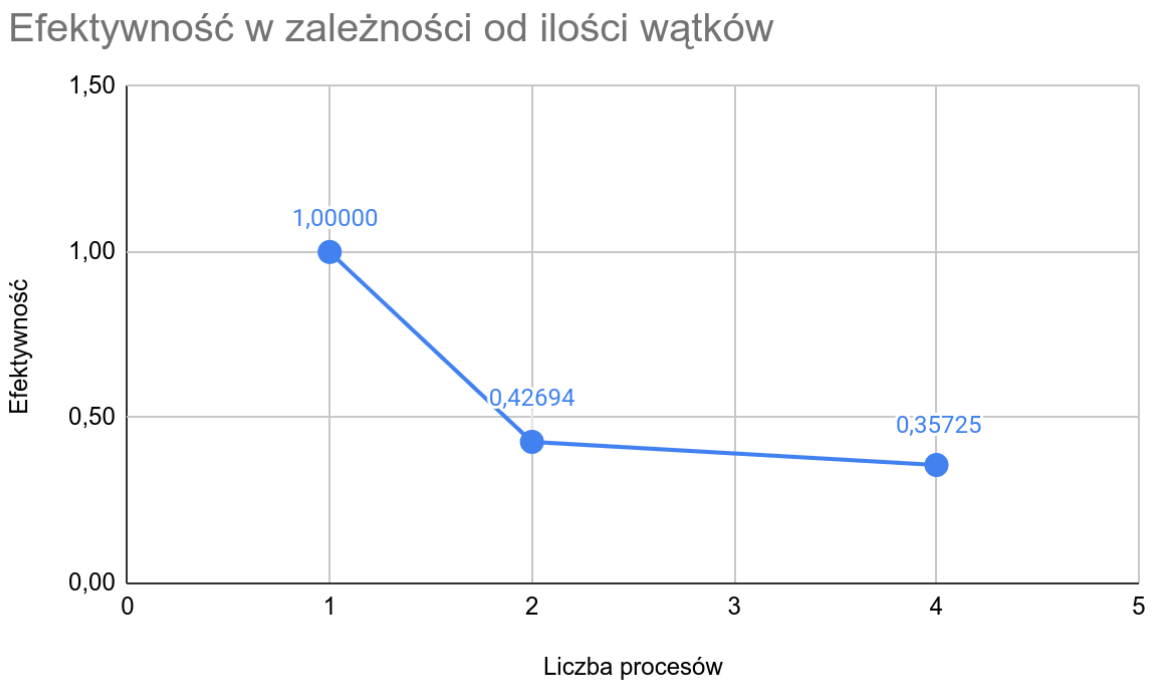
Ponownie obliczamy przyspieszenia

Liczba procesów	Przyspieszenie rzeczywiste	Przyspieszenie perfekcyjne
1	1,00000	1
2	0,85389	2
4	1,42901	4

Ponownie widać, że nie udało się nam osiągnąć najlepszego przyspieszenia, jest to najpewniej spowodowane wykorzystaniem innego sposobu zrównoleglania programu, wykorzystanie komunikacji między procesami najpewniej zajmuje dużo dodatkowego czasu, który powoduje to pogorszenie wyników.

Efektywność naszego programu wygląda następująco.

Liczba procesów	Efektywność
1	1,00000
2	0,42694
4	0,35725



Ponownie widać spory spadek efektywności naszego programu.

Skalowalność

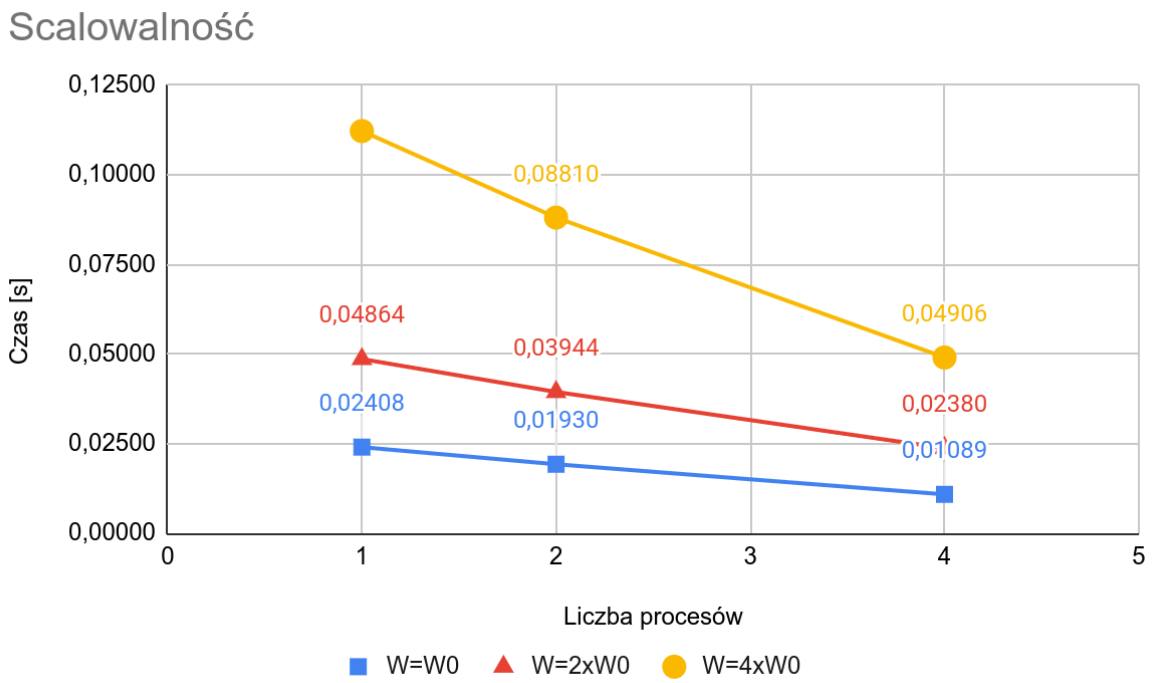
Ponownie wykorzystujemy program obliczający iloczyn macierz - wektor, który wykorzystuje MPI do zrównoleglenia programu. Tym razem jednak wraz ze wzrostem ilości procesów wzrasta nam proporcjonalnie wielkość problemu. Tak przykładowo dla 1 procesu rozmiar wynosi 3040, dla 2 procesów 4296, a dla 4 rozmiar wynosi 6072.
Następnie ponownie wykonaliśmy serię pomiarów czasów wykonania dla programu z flagą -O0 w celu uniknięcia optymalizacji.

Ilość procesów	W=W0	W=2xW0	W=4xW0
1	0,023912	0,049397	0,111589
1	0,023431	0,048819	0,115103
1	0,023863	0,048015	0,113598
1	0,025103	0,048329	0,108638
2	0,019044	0,039366	0,086727
2	0,01909	0,039134	0,088298
2	0,019247	0,039457	0,088449
2	0,019807	0,039804	0,088918
4	0,009781	0,02097	0,048035
4	0,009707	0,021255	0,049411
4	0,010499	0,021573	0,049691
4	0,013586	0,031407	0,049121

W tym przypadku ponownie możemy wykorzystać wszystkie uzyskane czasy przez brak odstających wyników. Z uzyskanych czasów obliczamy średnie wartości.

Ilość procesów	W=W0	W=2xW0	W=4xW0
1	0,02408	0,04864	0,11223
2	0,01930	0,03944	0,08810
4	0,01089	0,02380	0,04906

Zauważyć można, że w obrębie jednego rozmiaru mamy spadek czasu dla coraz to większej ilości procesów, jest to efekt który udało się nam zauważyć w poprzednich analizach.

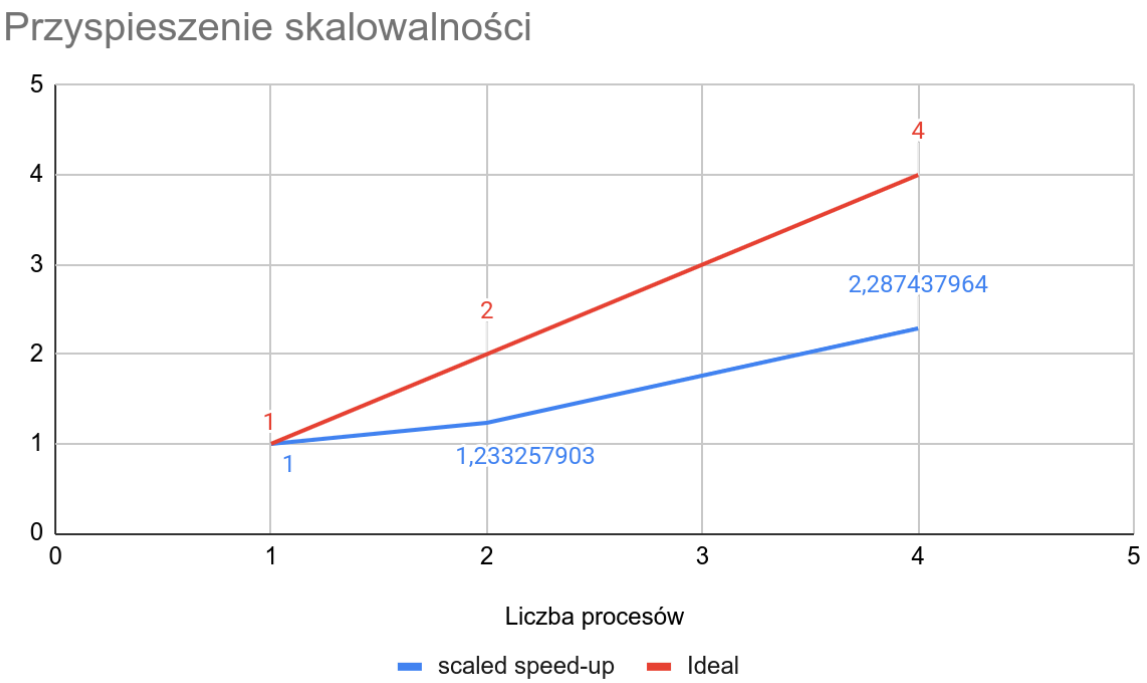


Tutaj będzie opis wykresu

Następnie liczymy przyspieszenie skalowalności, jest to stosunek średniego czasu wykonania programu dla 1 procesu z rozmiarem p*W0 do średniego czasu wykonania dla p procesów z rozmiarem p*W0.

Ilość procesów	scaled speed-up	Ideal
1	1	1
2	1,233257903	2
4	2,287437964	4

Ponownie dla porównania w trzeciej kolumnie znajdują się idealne wartości jakie chcielibyśmy osiągnąć, jednak jak widać nasze przyspieszenie nie jest idealne.



Jak widać z powyższych wykresów program jest skalowalny, ponieważ ze wzrostem rozmiaru oraz ilości procesów przyspieszenie skalowalności rośnie, jednak nie jest to przyspieszenie idealnie. Dla 2 oraz 4 wątków niestety dużo brakuje do idealnego przyspieszenia skalowalności.

Pomiar Gflops

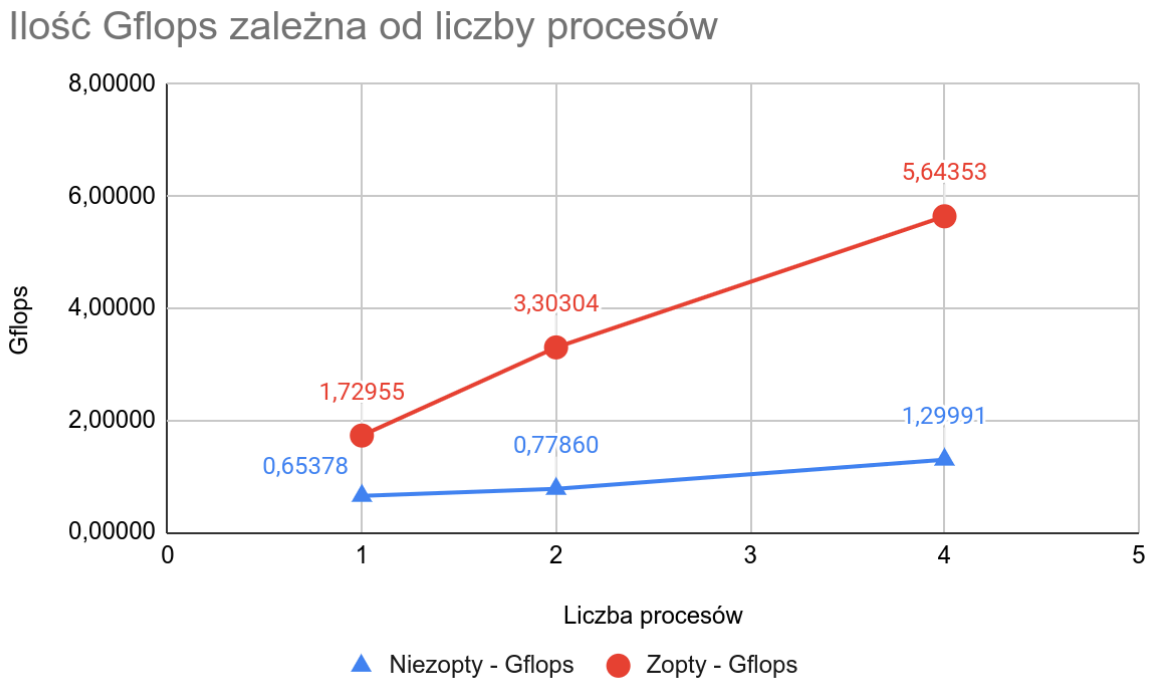
W kolejnym zadaniu chcemy obliczyć ilość wykonanych operacji zmiennoprzecinkowych na sekundę. Nasz program do obliczania iloczynu macierz - wektor zwraca informację o ilości wykonanych operacji zmiennoprzecinkowych. Program wykonujemy kilkakrotnie z różnymi ilościami procesów oraz w wersji zoptymalizowanej oraz bez optymalizacji. Następnie dla każdego wykonujemy kilka pomiarów czasów i liczymy wartości średnie.

Niezoptymalizowane		
LP	Czas	Ilość operacji
1	0,45115	294953472
2	0,37883	294953472
4	0,22690	294953472
Zoptymalizowane		
LP	Czas	Ilość operacji
1	0,17054	294953472
2	0,08930	294953472
4	0,05226	294953472

Jak widać ponownie dostajemy lepszy czas wykonania przy większej ilości wątków, dodatkowo można zauważyć różnicę w czasach pomiędzy wersją nie zoptymalizowaną a zoptymalizowaną.
Następnie na podstawie tych danych obliczamy ilość flopsów dla każdego wariantu uruchomienia programu, jednostkę podajemy w Gflopsach dla lepszej czytelności.

Liczba procesów	Niezopty - Gflops	Zopty - Gflops
1	0,65378	1,72955
2	0,77860	3,30304
4	1,29991	5,64353

Jak można zauważyć wraz ze wzrostem ilości procesów nasza ilość Gflopsów również rośnie.



Ponieważ nasz kod jest w stanie szybciej się wykonać przy większej ilości procesów, dlatego również jest w stanie wykonać więcej operacji zmiennoprzecinkowych w tym samym czasie. Dobrze to pokazuje powyższy wykres.
Dodatkowo można zauważyć różnicę w ilości wykonanych operacji dla wersji zoptymalizowane, a nie zoptymalizowanej, Dla zoptymalizowanej dostajemy wielokrotnie większą ilość Gflops, największą różnicę dostajemy przy 4 procesach, prawie 5 krotny wzrost.

Wnioski

Z naszej analizy udało się wywnioskować, że wszystkie kody działają efektywniej w wersjach równoległych, jest to oczywiste w przypadku dużych problemów i warte uwagi. Dla małych problemów często zrównoleglenie będzie niepotrzebne przez dodatkowy narzut tworzenia procesów/wątków oraz komunikacji między nimi.
Dodatkowo w pierwszym porównaniu warto zauważyć dużą różnicę między zrównolegleniem programów przez OpenMP a MPI, pomimo tego, że są to dwa różne programy rozwiązujące różne problemy można zauważyć dużą różnicę w przyspieszeniach tych mechanizmów. Dla OpenMP uzyskaliśmy bardzo zbliżony do ideału, a w przypadku MPI niestety nie jest aż tak bliska ideałowi, jednak dalej jest bardziej korzystna. Korzyści zrównoleglenia widać przede wszystkim na wykresach efektywności, oba mechanizmy były bardziej efektywne dla wersji zrównoleglonej.
W przypadku programu MPI udało się nam również zauważyć, że program w miarę dobrze się skaluje przy wzroście wielkości problemu, ponownie do idealnej skalowalności brakowało trochę, ale dalej efekty były zadowalające.
Dla ostatniego przykładu z pomiarem Gflops mogliśmy również zauważyć jak optymalizacja przy kompilacji wpływa na ilość możliwych wykonanych operacji zmiennoprzecinkowych. Efekty są bardzo zadowalające, ponieważ dostaliśmy kilkukrotny wzrost wartości.