Imię i nazwisko: **Anna Dybel**	Kierunek i grupa: **Inżynieria Obliczeniowa gr. 1**	Data: **27.01.2021 r.**
Sprawozdanie z lab. 14	Przedmiot: **Przetwarzanie współbieżne. Programowanie równoległe i rozproszone.**	

1. Cel:

Doskonalenie umiejętności analizy wydajności programów równoległych.

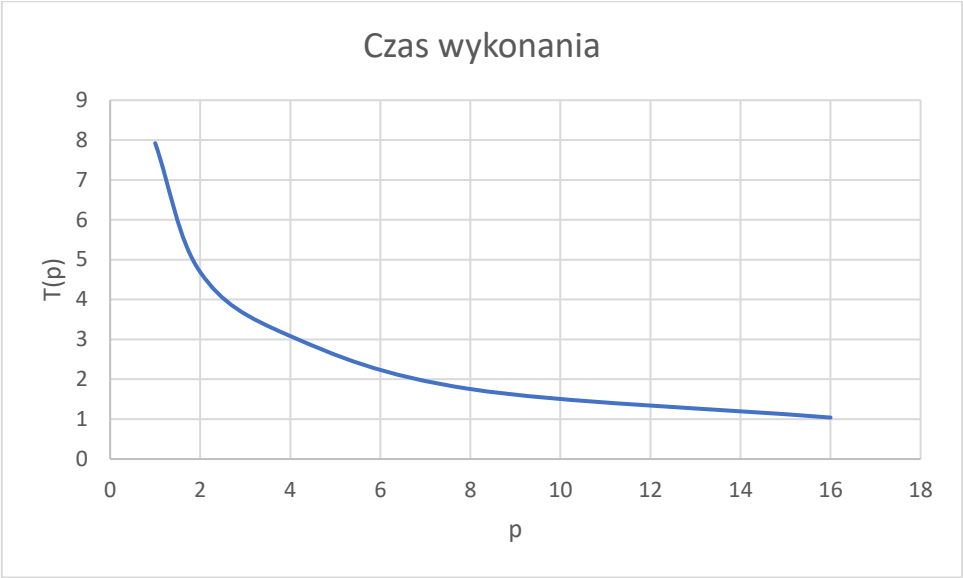
2. W ramach zajęć zrealizowałam następujące kroki:
   1. Utworzyłam katalog lab_15 do którego pobrałam i rozpakowałam następujące paczki MPI_calka.tgz i mat_vec_row_omp.tgz.

   2. Następnie przystąpiłam do przeprowadzania testów, które wykonałam na serwerze ESTERA. Dla każdego pomiaru przeprowadziłam trzy próby i wybrałam czas najmniejszy.

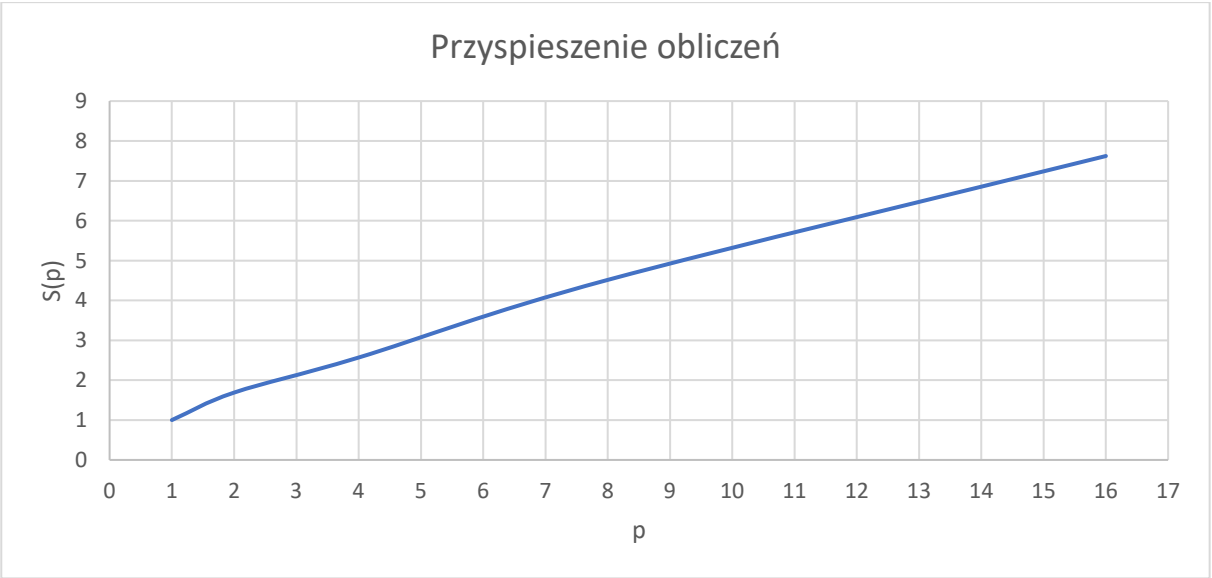
   3. Pierwsze testy wydajności przeprowadziłam dla programu liczącego całkę. Pomiary czasu wykonałam dla 1, 2, 4, 8 i 16 procesów oraz stałego rozmiaru zadania. Czas jest potrzebny do wyznaczenia miar względnych takich jak przyspieszenie obliczeń i efektywność zrównoleglenia.

liczba procesów p	rozmiar zadania	czas wykonania T	S(p)	E(p) [%]
1	479001600	7,923105	1	100
2	479001600	4,689906	1,689395267	84,46976336
4	479001600	3,084291	2,568857802	64,22144506
8	479001600	1,754545	4,515760496	56,4470062
16	479001600	1,039426	7,622577269	47,64110793

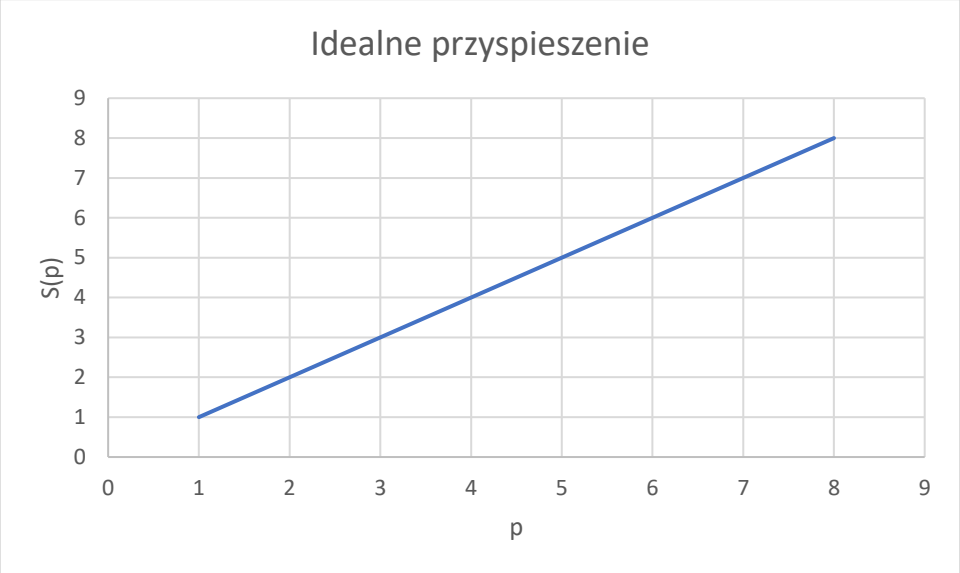
Przyspieszenie zostało wyliczone z następującego wzoru $\mathbf{S(p)} = \frac{\mathbf{T_{Sekw}}}{\mathbf{T(p)}} = \frac{\mathbf{T(1)}}{\mathbf{T(p)}}$, natomiast efektywność $\mathbf{E(p)} = \frac{\mathbf{S(p)}}{\mathbf{p}} * 100\%$**.** Ponieważ nie mamy programu napisanego sekwencyjnie, więc w tym celu aby otrzymać czas dla obliczeń sekwencyjnych wystarczyło uruchomić program dla jednego procesu. Dla danych zamieszczonych w tabelce otrzymałam następujące wykresy.

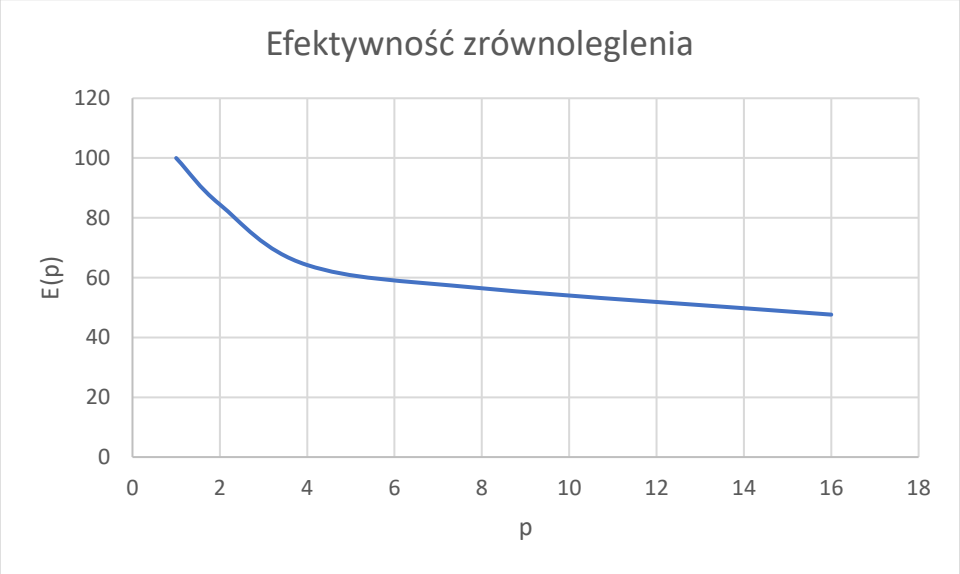


Powyższy wykres przedstawia czas wykonania programu w zależności od użytej ilości procesów. Intuicyjnie czas wykonania programu powinien maleć wraz z zwiększaniem ilości procesów. Dla dwóch procesów czas wykonania powinien być dwa razy mniejszy niż sekwencyjny program, dla czterech procesów cztery razy szybszy itp. Jak widać, krzywa na wykresie wraz ze wzrostem procesów maleje.



Powyższy wykres przedstawia przyspieszenie obliczeń, czyli stosunek czasu wykonania programu równoległego do najlepszego programu sekwencyjnego czyli takiego, który osiąga minimalny czas wykonania. Idealne przyspieszenie przedstawiałby wykres $S(p) = p$, czyli uzyskanie liniowego przyspieszenia (Wykres znajduje się poniżej). Jak widać wykres jest zbliżony do wykresu idealnego przyspieszenia ale krzywa znajduję się poniżej krzywej idealnego przyspieszenia. To oznacza, że dla danego procesu $S(p) < p$ (często uzyskuję się takie przyspieszenie). Przyczyną tego może być wymiana danych miedzy procesami lub synchronizacja ich pracy co może prowadzić do niezrównoważonego obciążenia pracy. Intuicyjnie przyspieszenie powinno być większe od 1 dla obliczeń równoległych.





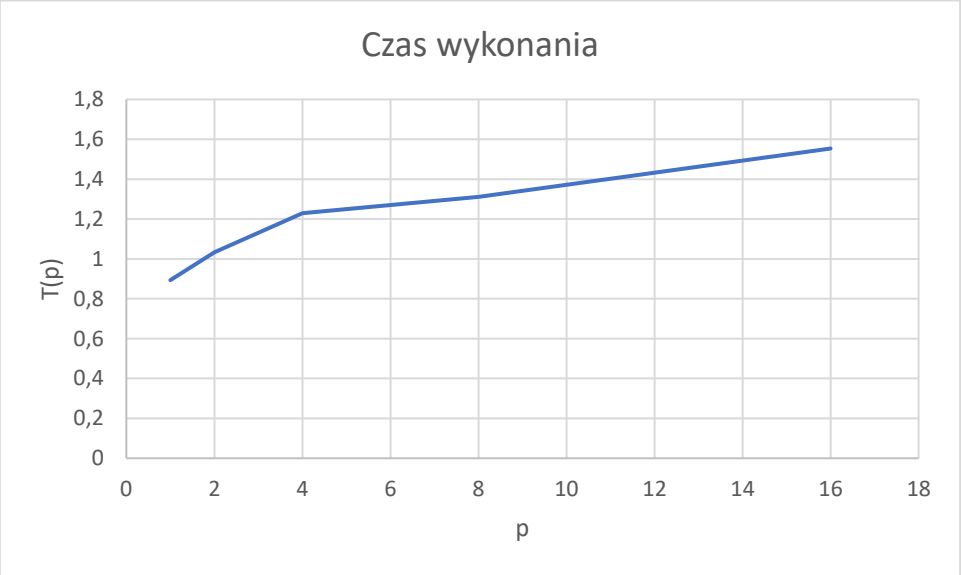
Powyższy wykres przedstawia efektywność zrównoleglenia. Teoretycznie idealny wykres efektowności przedstawiałaby krzywą E(p) = 100%, która nie spadałam by poniżej pewnej wielkości czyli efektywność dla każdego procesu wynosiłaby 100%. Taka efektywność oznaczałaby brak bezczynności procesów i brak komunikacji między nimi. Jak widzimy, na wykresie krzywa maleje wraz ze zwiększeniem procesów co jest właśnie spowodowane zwiększaniem się kosztów komunikacji miedzy procesami.

4. Następnie przesyłam do testów skalowalności (Program jest skalowalny wtedy gdy jego efektywność pozostaje na takim samym poziomie przez jednoczesne zwiększanie procesów i rozmiaru obliczeń.) w sensie słabym czyli stały rozmiar zadania dla danego procesu. Przeprowadziłam testy dla 1, 2, 4, 8, 16 procesów przy czym rozmiar zadania dla pierwszego procesu ustawiłam na wartość 43545600 a dla kolejnych procesów rozmiar zadania jest proporcjonalny od ilości procesów czyli p*43545600.

liczba procesów p	rozmiar zadania	czas wykonania
1	43545600	0,893371
2	87091200	1,033564
4	174182400	1,23011
8	348364800	1,311383

16	696729600	1,553755

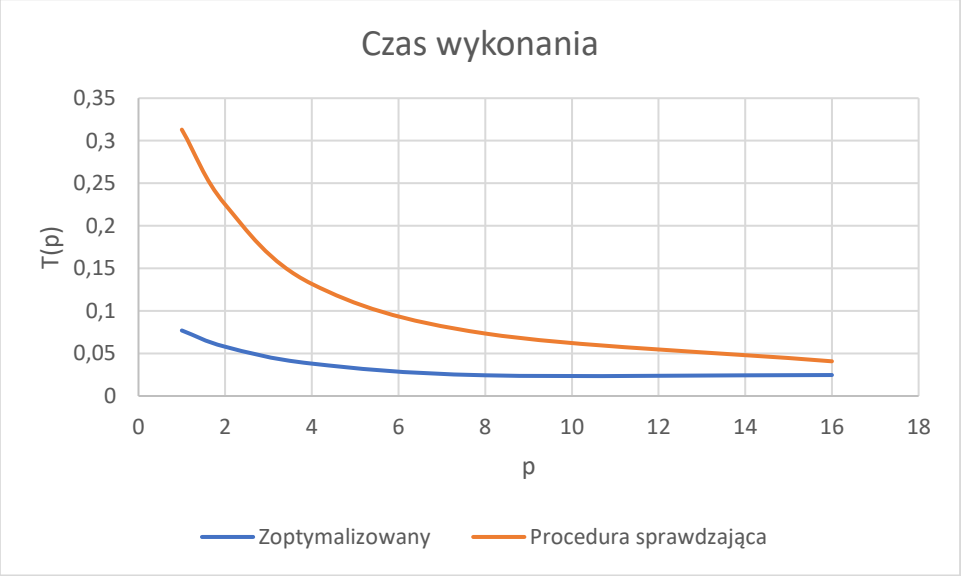
Intuicyjnie, jeśli czas dla jednego procesu (czyli obliczeń sekwencyjnych) i rozmiaru zadania wynosi T to dla 2*rozmiar_zadania na jednym procesie czas powinien wyjść 2*T. Z tego powodu uruchomienie obliczeń dla 2*rozmiar_zadania na dwóch procesach powinien być dwa razy krótszy od czasu sekwencyjnego czyli (2*T)/2 = T. Biorąc pod uwagę wcześniej wspomnianą zależność otrzymany wykres powinien przedstawiać krzywą, która jest stałą **T(p*rozmiar_zadania, p) = T(rozmiar_zadania, 1)**.



Patrząc na otrzymany wykres z danych przedstawionych w tabelce 2, widać że nie wygląda on tak jak opisałam to wcześniej i krzywa nie pokrywa się z krzywą **T(p*rozmiar_zadania, p) = T(rozmiar_zadania, 1)**. Czasy czyli koszt obliczeń wraz ze wzrostem procesów i rozmiaru zadania rosną, Jest to spowodowane architekturą sprzętu i tym że częstotliwość przy jednym procesie jest większa przez to zadanie wykonuję się szybciej niż przy większej ilość procesów gdzie częstotliwość jest już mniejsza.

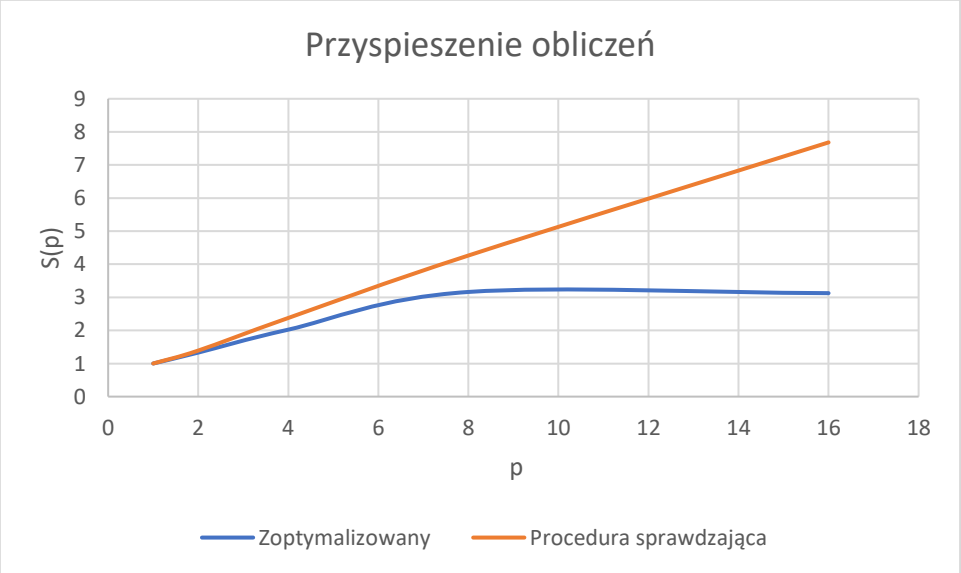
5. Przystąpiłam do przeprowadzenia testów dla programu liczącego iloczyn macierz wektor dla obu wersji programu czyli maksymalnie zoptymalizowanej (opcja -O3) oraz niezoptymalizowanej procedury sprawdzającej. Test wykonałam dla 1, 2, 4, 8, 16 wątków po kilka prób, wybierając najmniejsze czasy.

liczba procesów p	T(p) zoptymalizowany	T(p) procedura sprawdzająca
1	0,077003	0,313072
2	0,057916	0,225016
4	0,038062	0,131729
8	0,02435	0,073424
16	0,024634	0,040754



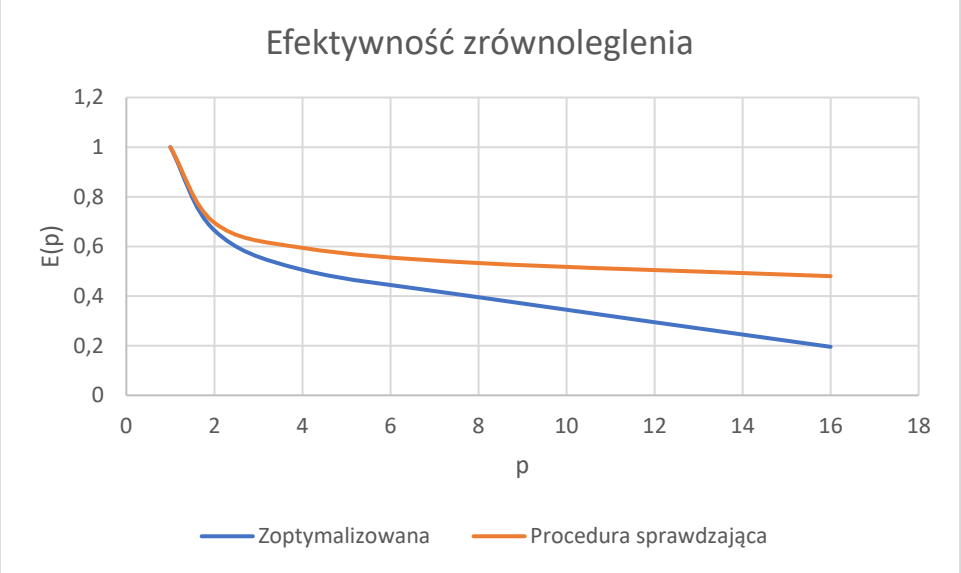
Jak widać krzywa dla programu w wersji nie zoptymalizowanej przypomina krzywą z wykresu czasu wykonania dla programu liczącego całkę. Wyraźnie widać równice miedzy wykonaniem sekwencyjnym a równoległym. Jeśli chodzi o krzywą dla programu silnie zoptymalizowanego to można zaobserwować że obliczenia równoległe wcale bardzo nie skracają czasu wykonania programu a wręcz dla 8 i 16 wątków czas jest ten taki sam. Można z tego wywnioskować że zrównoleglanie programów zoptymalizowanych może być nie opłacalne ponieważ czas sekwencyjny jest dużo lepszy niż programu niezoptymalizowanego sekwencyjnego a uruchamianie takiego programu na wątkach może nie przynieść żadnych rezultatów a wręcz nawet pogorszyć czas w zależności od tego czy będzie występować synchronizacja miedzy wątkami i komunikacja.

liczba procesów p	S(p) zoptymalizowany	S(p) procedura sprawdzająca
1	1	1
2	1,329563506	1,391332172
4	2,023093899	2,376636883
8	3,162340862	4,263891915
16	3,125882926	7,681994405



Podobne wnioski można także zauważyć analizując powyższy wykres przyspieszenia obliczeń. Dla procedury sprawdzającej krzywa jest zbliżona do krzywej idealnego przyspieszenia S(p) < p i widać skrócenie czasu wykonywania się obliczeń. Natomiast dla wersji zoptymalizowanej widać że dla 2, 4, 8 wątków jest lekkie przyspieszenie ale dla 16 wątków przyspieszenie jest mniejsze niż dla 8 wątków i to przy takim samym rozmiarze zadania.

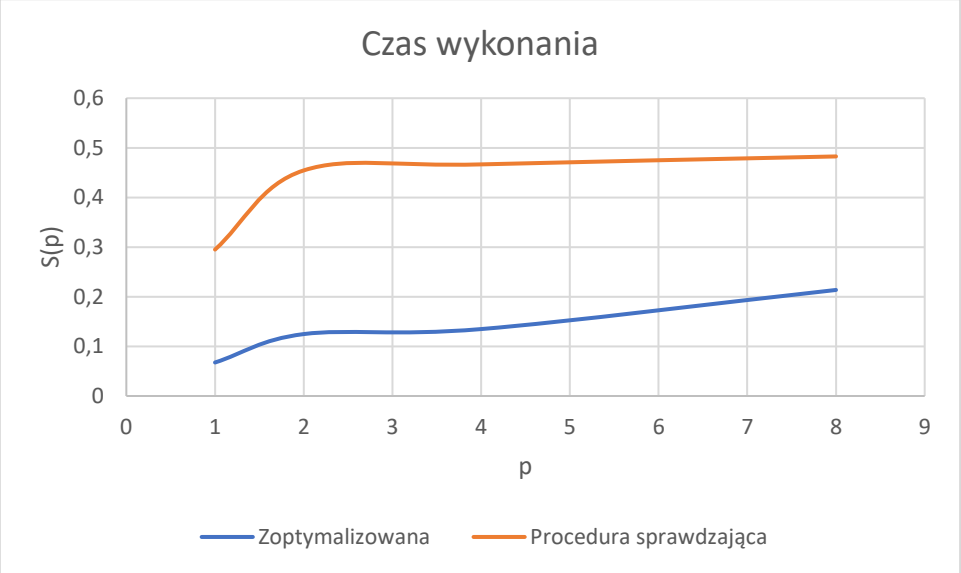
liczba procesów p	E(p) macierz wektor	E(p) procedura sprawdzająca
1	1	1
2	0,664781753	0,695666086
4	0,505773475	0,594159221
8	0,395292608	0,532986489
16	0,195367683	0,48012465



Efektywność wersji zoptymalizowanej jest także dużo niższa

6. Następnie przeszłam do testów skalowalności programu liczącego iloczyn macierz wektor. Tym razem testy przeprowadziłam dla 1, 2, 4, 8 procesorów z tego powodu że dla rozmiaru zadania 4*WYMIAR program zgłaszał przekroczenia rozmiaru pamięci, ponieważ programie są wykorzystywane statyczne tablice. Analogicznie jak w teście skalowalności dla programu liczącego całkę ustawiłam rozmiar zadania czyli WYMIAR na wartość 10000 dla jednego wątku. Następnie dla wraz ze wzrostem wątków zwiększam rozmiar zadania w następującej sekwencji sqrt(p)*WYMIAR. Intuicyjnie, tak samo jak przy programie liczącym całkę powinna wyjść krzywa **T(sqrt(p)*WYMIAR, p) = T(WYMIAR, 1)**, lecz ze względów architektury krzywa może wyjść inna.

liczba procesów p	macierz-wektor	procedura sprawdzająca	WYMIAR
1	0,067526	0,295053	10000
2	0,124743	0,454496	14142
4	0,134853	0,466654	20000
8	0,213639	0,482668	28284



Czas obliczeń jednego wątku dla procedury sprawdzającej różni się od czasów pozostałych wątków co jest spowodowane architekturą. Wyniki dla 2, 8 i 16 wątków jest zbliżony do siebie i krzywa tym miejscu i jest prawie stała.

3. Wnioski

Celem zajęć było przeprowadzenie testów wydajności programów równoległych. Jest to bardzo istotne, ponieważ programy równoległe powinny dawać korzyści, których nie dają programy sekwencyjne czyli przyspieszenie obliczeń i maksymalne wykorzystanie dostępnej architektury. Poprzez testy można ocenić, który z algorytmów jest lepszy optymalniejszy oraz jak dane rozwianie jest przydatne lub nie przydatne przy danej technologii. Ocena programów równoległych jest bardziej skomplikowana niż ocena programów sekwencyjnych. Do oceny obliczeń równoległych wykorzystaliśmy miary względne takie jak przyspieszenie obliczeń i efektywność zrównoleglenia. Testy zostały wykonane dla tego samego zadania przy różnej ilości wątków lub procesów. Obserwując uzyskane wykresy można zauważyć, że czas wykonywania obliczeń dla tego samego zadania maleje (tak jak można by było przypuszczać) a pozostałe wykresy są zbliżone do wykresów idealnych na co wpływa dużo czynników. Takimi czynnikami może być komunikacja i synchronizacja miedzy wątkami i procesami, która istotnie wpływa na zmniejszanie się efektywności. Kolejnymi czynnikami może być realizacja operacji sekwencyjnych, których nie da się zrównoleglić. Warto też podkreślić to że programy które były testowane różniły się miedzy sobą. Program do obliczania iloczynu macierzy wektor musiała mieć dostęp do pamięć co mogło wpłynąć na czas obliczeń jeśli np. dane nie były w pamięci podręcznej. Dodatkowo przy dużej ilości wątków mógł się pojawić problem z przepustowością magistrali co znacznie mogło spowolnić działanie programu. Częstotliwość także mogła wpłynąć na uzyskane wyniki. Jak widać uzyskane wyniki będą się różnić w zależności od rozwiązań i architektury i może się zdarzyć że program równoległy wydajny na jednym urządzeniu będzie wydajny na innym. Oprócz wiedzy i umiejętności tworzenia programów równoległych należy też umieć je testować i mieć świadomość różnić miedzy technologiami.