Metody programowania równoległego

Patryk Wojtyczek

Kod programów jak i skrypty do uruchomiania programów zamieściłem w zipie.

Część 1 - Komunikacja P2P

Programy do pomiaru przepustowości zostały napisane w języku `c` i były wykonywane na klastrze `vnode-*.dydaktyka.icsr.agh.edu.pl`. Celem zadania jest zmierzenie wartości opóźnienia i charakterystyki przepustowości połączeń w klastrze.

Funkcje wykorzystane do komunikacji:

- `ssend` - Synchronous blocking send.

Blokuje dopóki odbiorca nie odebrał wiadomości. Po wykonaniu tej funkcji można bezpiecznie korzystać z przekazanego bufora.

- `ibsend` - Asynchronous non-blocking send.

Funkcja ta stworzy kopię przekazanych danych i wyśle ją w późniejszym czasie. Funkcja nie blokuje wywołującego, i po jej wywołaniu nie można modyfikować bufora gdyż mógł on jeszcze nie zostać skopiowany. Do sprawdzenia czy bufora można użyć służy funkcja `MPI_Test` lub `MPI_Wait`.

Pomiary wykonywałem w następujący sposób. Z góry ustaliłem ilość danych do przesłania - 100MB. Dla każdej rozważanej wielkości wiadomości obliczałem ile wiadomości trzeba wysłać aby przesłać ustaloną wyżej ilość danych. Na podstawie czasu potrzebnego na wykonanie tranferu tych danych obliczałem przepustowość.

Jako MB - przyjąłem (być może zaskakująco) 10^6B zgodnie z układem SI.

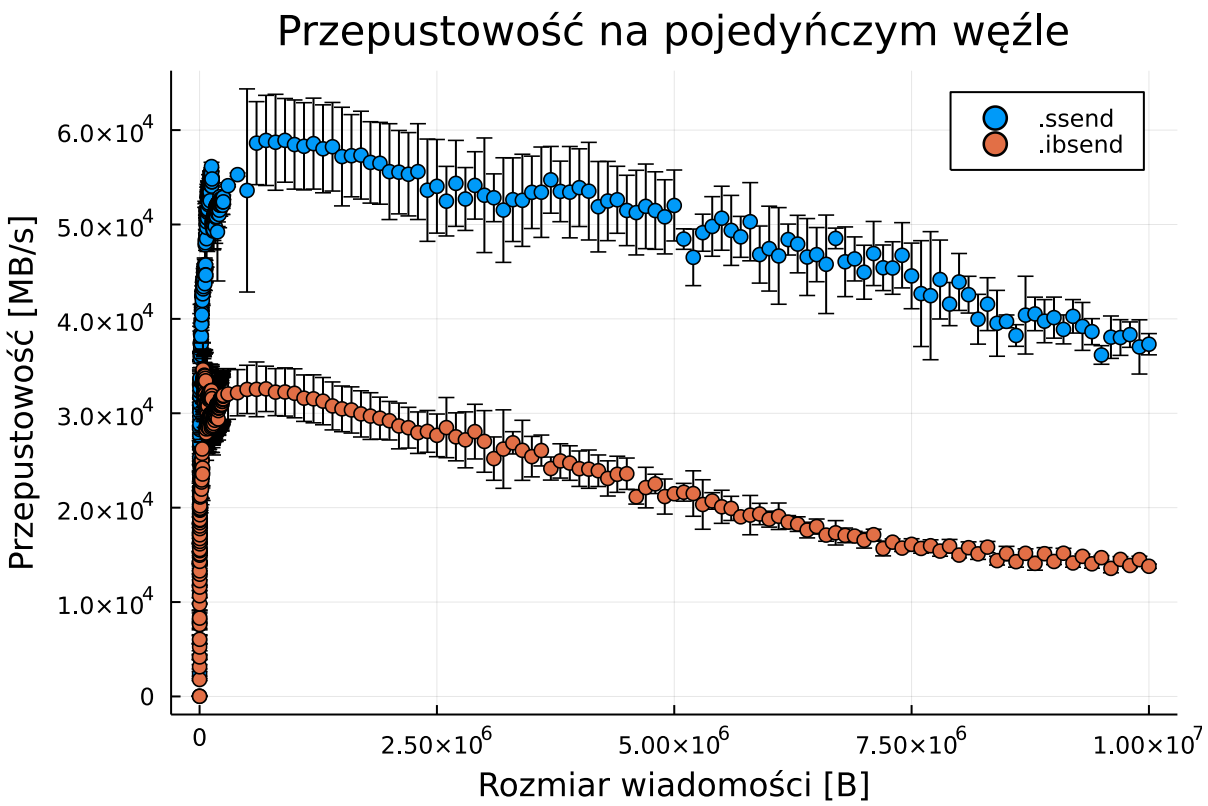
Zmierzyłem wiadomości w zakresie 1B-10MB, przy czym większość zmierzonych rozmiarów (około 300) mieści się poniżej 250kB, pozostałych jest około 100. Dla każdej wielkości wiadomości powtórzyłem pomiar trzykrotnie.

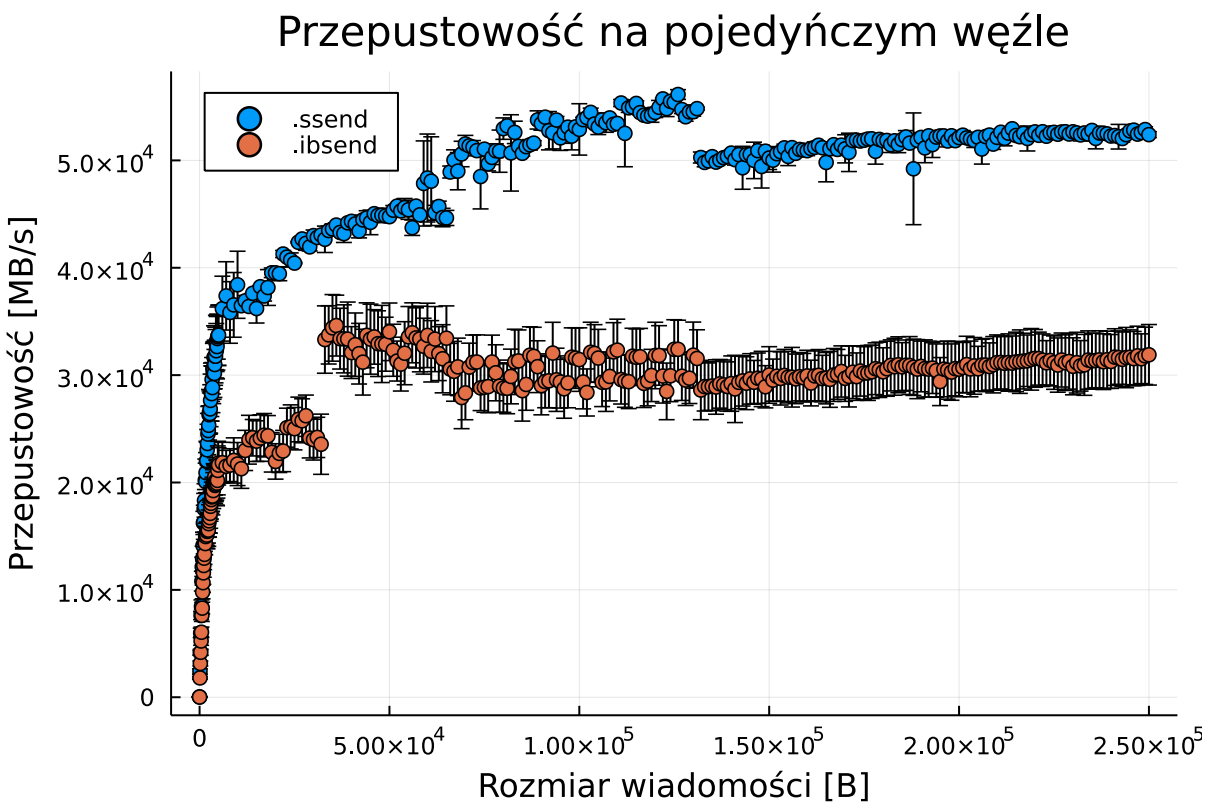
Przepustowość na pojedyńczym węźle

Na pojedyńczym węźle przepustowość P2P jest olbrzymia. Największy throughput uzyskujemy dla wiadomości o rozmiarze 100kB - 1MB, później throughput zaczyna spadać. Dla bardzo małych wiadmości throughput jest bardzo niski gdyż narzut ze względu na komunikację między procesami jest znacznie większy od narzutu ze względu na przenoszenie danych.

`Ssend` osiąga większy throughput od ibsend ze względu na fakt, że ssend nie robi dodatkowych kopii i to koszt tego kopiowania widzimy na wykresie. Gdyby program na którym testujemy nie działał na zasadzie `wyślij-ping, odbierz-ping, wyślij-pong, odbierz-pong` tylko raczej `wyślij kilkanaście wiadomości i jakiś czas później sprawdź czy zostały wysłane` to myślę, że buforowanie wiadomości miałoby pozytywny wpływ na performance.

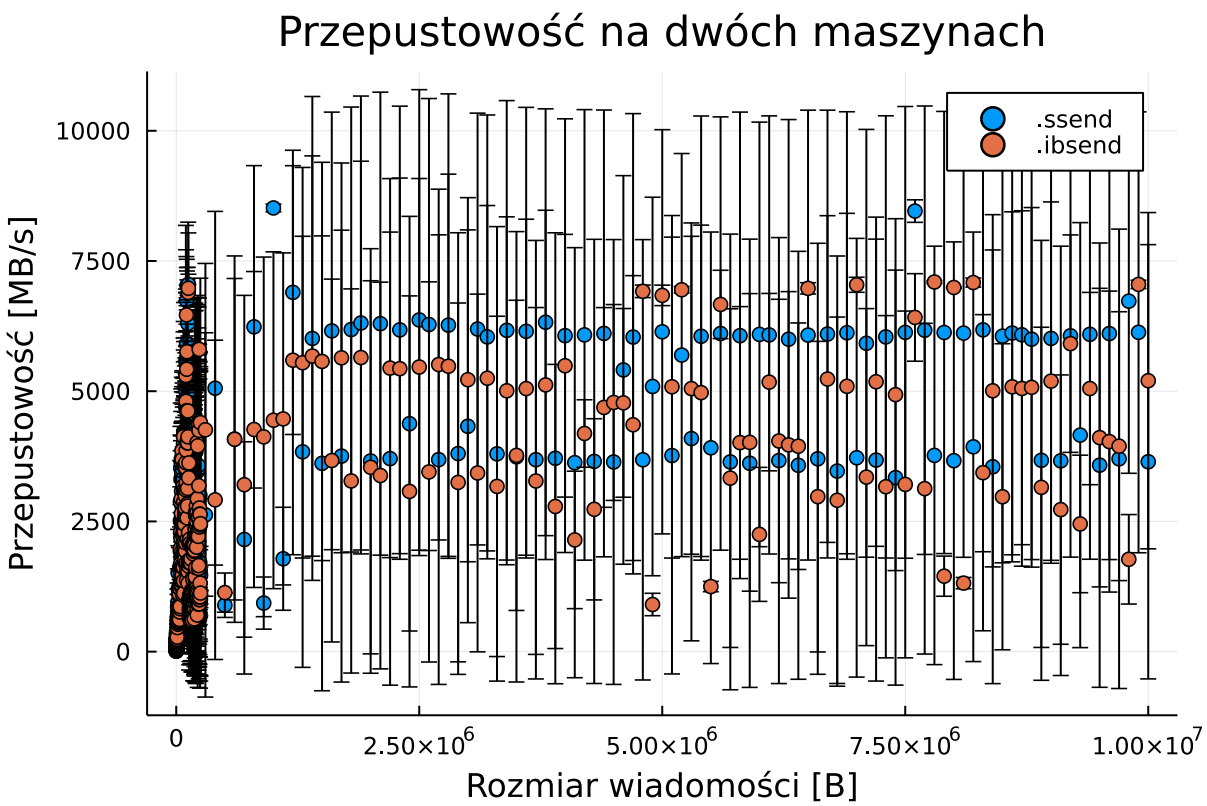
Poniżej przedstawiłem wykresy przepustowości dla wszystkich przetestowanych rozmiarów i tylko dla mniejszych rozmiarów wiadomości.

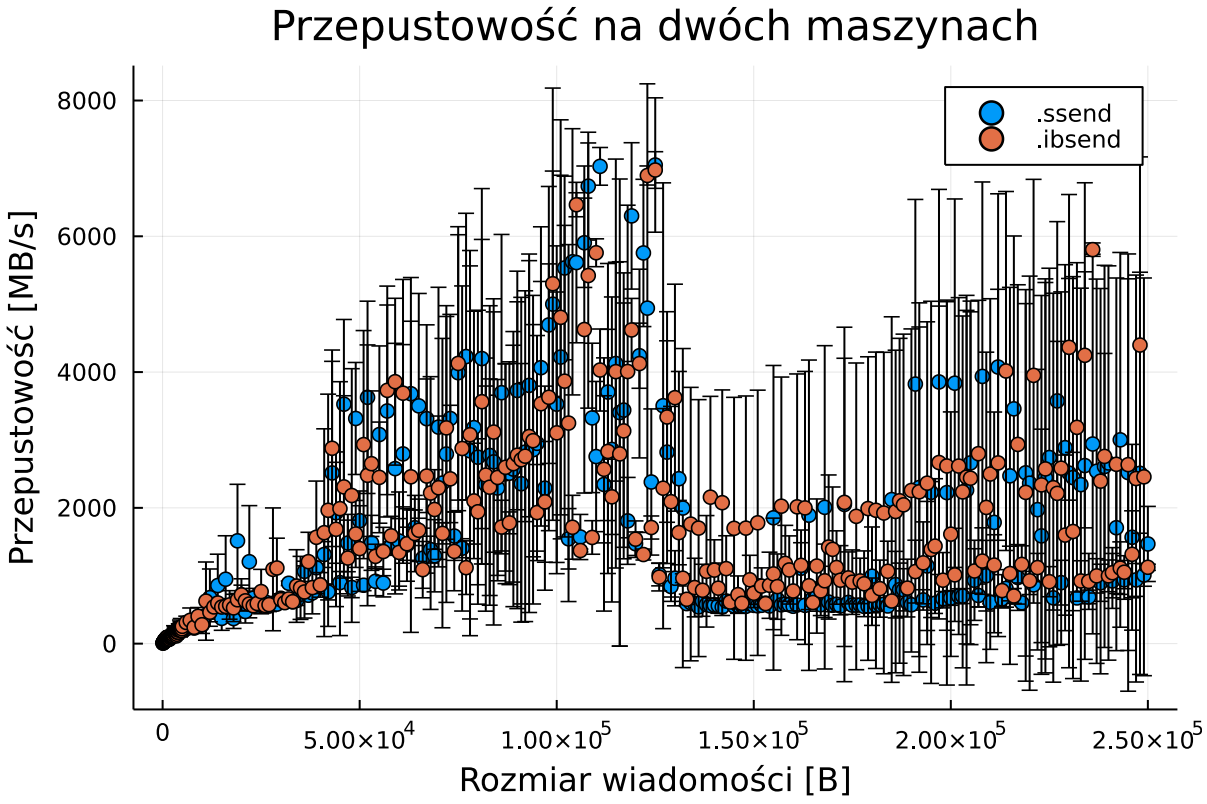




Przepustowość na dwóch węzłach

Tutaj wykonane pomiary były obarczone bardzo dużą wariancją. Wykres wygląda o tyle podobnie, że znowu małe rozmiary wiadmości dają niską przepustowość i wraz ze wzrostem rozmiaru wiadomości osiągamy coraz lepszą przepustowość. Mimo wszystko widać, że przepustowość spadła conajmniej 10-krotnie ze względu na komunikację przez sieć. Nie widać tutaj też trendu: im większy rozmiar wiadomości tym throughput zaczyna spadać. Różnice pomiędzy funkcjami wydają się być pomijalne, głównie ze względu na dominujący narzut w samej komunikacji. Widać też ciekawy spadek przepustowości w okolicach $1.25 \cdot 10^5 kB$ choć ciężko powiedzieć czym jest spowodowany.


Przepustowość na dwóch maszynach
Przepustowość [MB/s]
10000
7500
5000
2500
0
.ssend
.ibsend
0
2.50×10^6
5.00×10^6
7.50×10^6
1.00×10^7
Rozmiar wiadomości [B]


Przepustowość na dwóch maszynach
Przepustowość [MB/s]
8000
6000
4000
2000
0
.ssend
.ibsend
0
5.00×10^4
1.00×10^5
1.50×10^5
2.00×10^5
2.50×10^5
Rozmiar wiadomości [B]

Opóźnienie

Na podstawie zebranych danych można oszacować opóźnienie (czyli czas na wysłanie 1B) w komunikacji.

W przypadku funkcji `ssend` jest to:

- jeden węzeł: 7.0e-5 ± 7.5e-5s
- dwa węzły: 0.03837 ± 0.00027s

W przypadku funkcji `ibsend` jest to:

- jeden węzeł: 7.0e-5 ± 7.4e-5s
- dwa węzły: 0.038 ± 0.0074s

Część 2 - Badanie efektywności programu równoległego

Program do szacowania liczby pi metodą monte carlo został napisany w języku `c`. Wyniki zostały zebrane z wielokrotnego uruchomiania go na prometeuszu. Program posiada znikomą część sekwencyjną (faza reduce, gdy czekamy na wyniki od pozostałych węzłów) stąd możemy go porównywać do programu idealnie równoległego.

Pomiary

Pomiary wykonałem dla pięciu liczb punktów:

- π s1: 20000000 = $2 \cdot 10^7$
- π s2: 40000000 = $4 \cdot 10^7$ - mniej niż sekunda dla jednego węzła.
- π s3: 400000000 = $4 \cdot 10^8$
- π s4: 4000000000 = $4 \cdot 10^9$
- π s5: 20000000000 = $2 \cdot 10^{10}$ - ponad minuta ze wszystkimi węzłami.

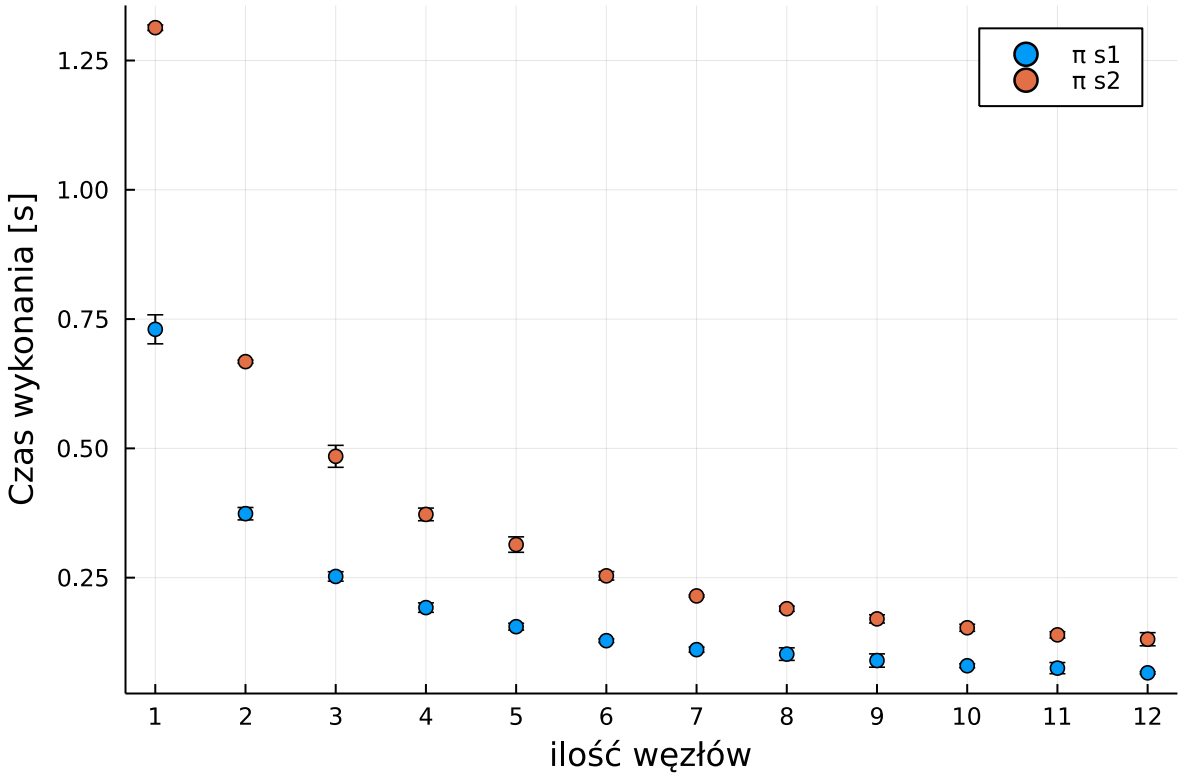
Pomiary zostały powtórzone wielokrotnie (między 10-20 razy) oprócz grupy π s5 gdzie udało mi się je wykonać tylko raz i rozpocząć drugi pomiar.

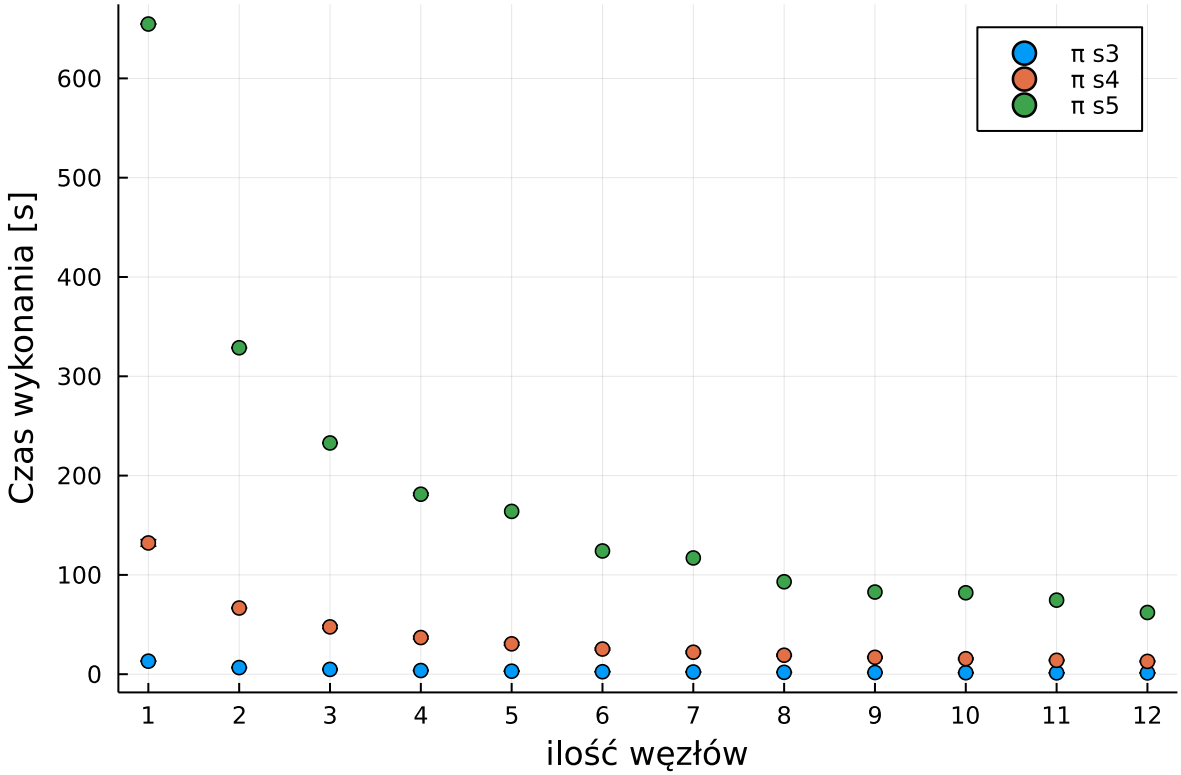
Większość wykresów jest podzielona na dwa ze względu na fakt, że wykresy wyglądają dość nieczytelnie z pięcioma grupami.

Czas wykonania

Widzimy, że czas wykonania w zależności od liczby węzłów przypomina funkcję $\frac{1}{x}$. Nie obserwujemy tutaj sytuacji w której dodawanie węzłów nie przyspiesza obliczenia. Można się było odrobinę spodziewać jakiegoś spadku przy 8 węzłach ze względu na to, że tyle mieści się w jednym sockecie, a później mamy droższą komunikację między socketami natomiast w tym problemie sama przepustowość komunikacji nie jest tak istotna bo nie przesyłamy dużych ilości danych.

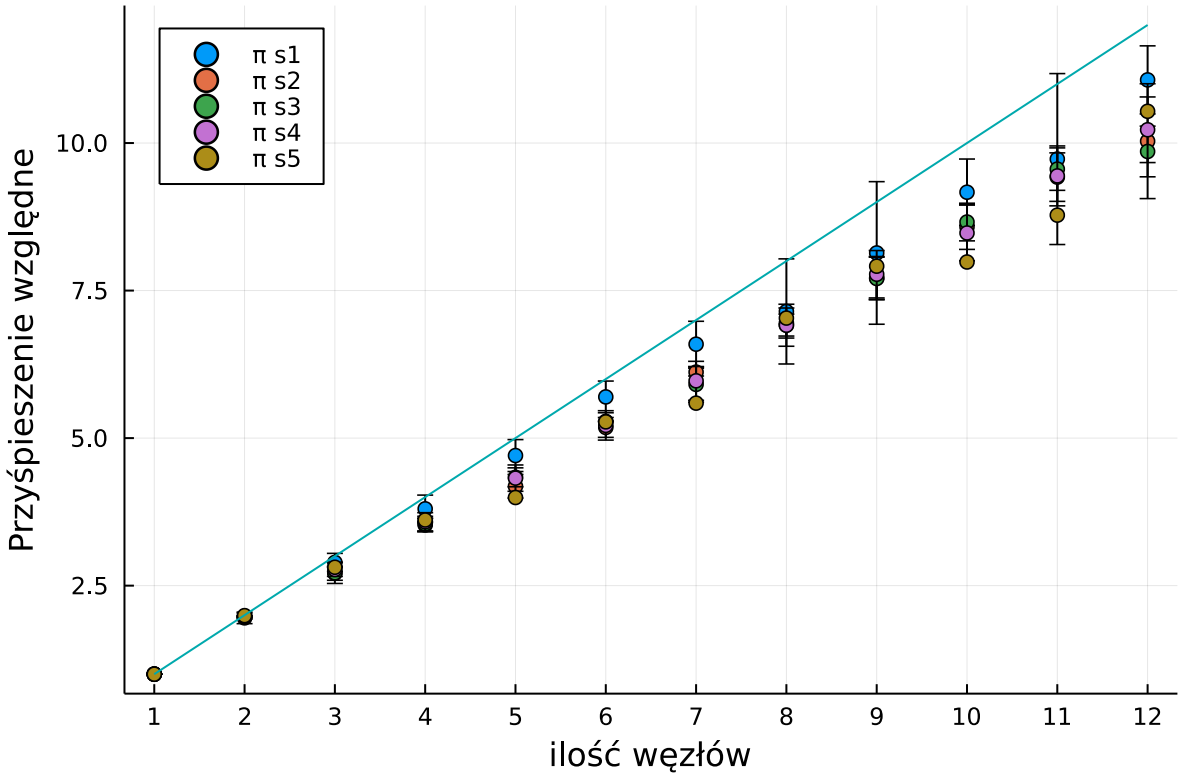
Przy większej ilości węzłów zysk z każdego następnego węzła nie jest aż tak zauważalny jak dla kilku węzłów.





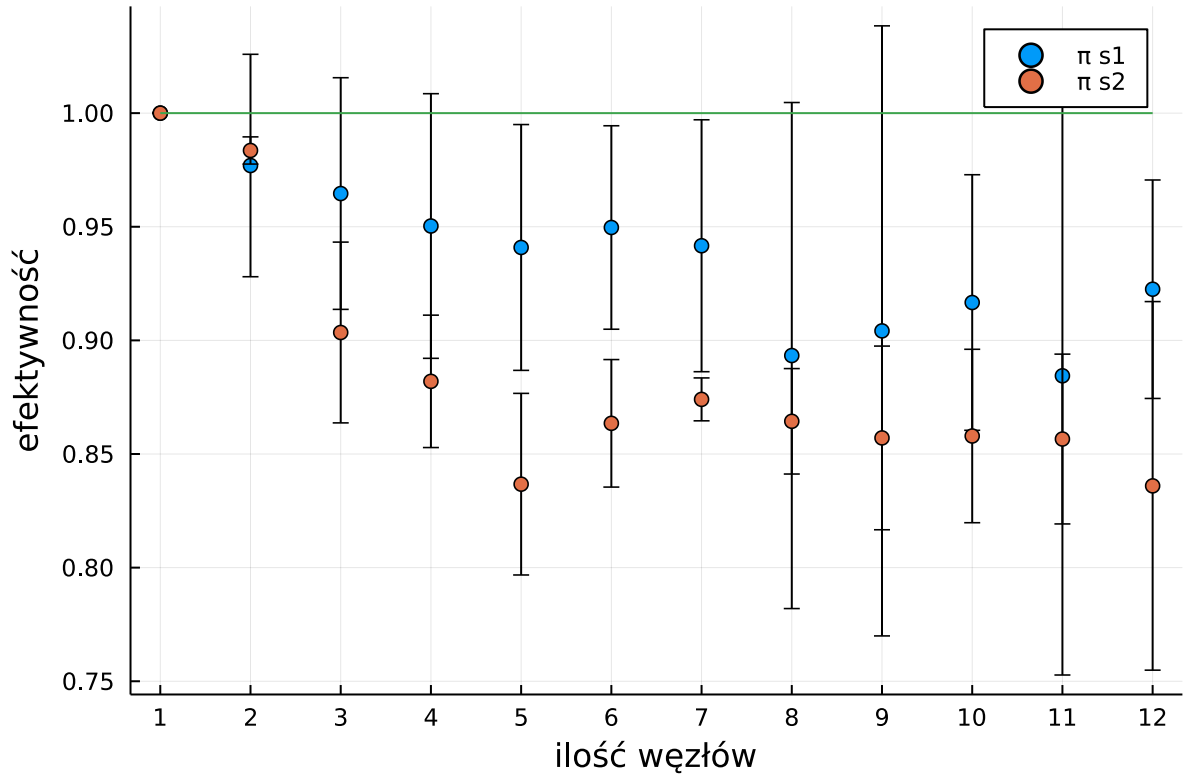
Przyspieszenie względne

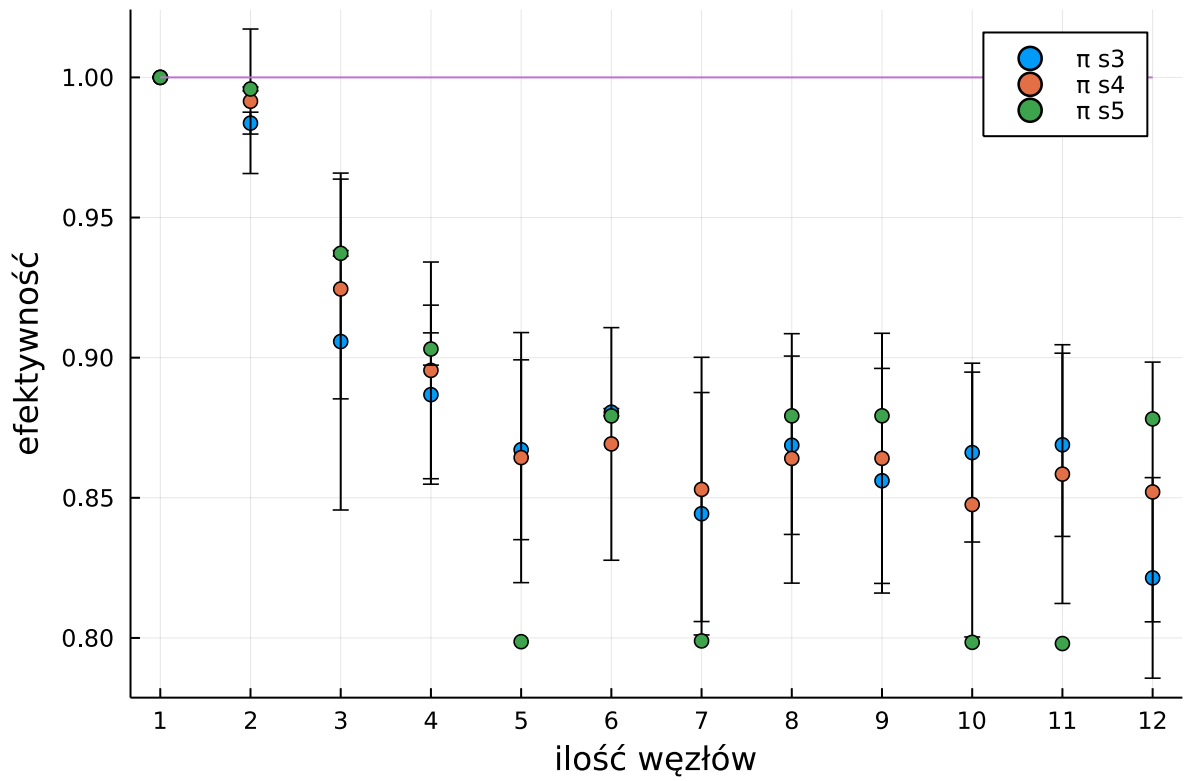
Widać, że o ile na początku trzymamy się idealne przyspieszenia to wraz z dodawniem nowych węzłow odchodzimy od tej linii. Jest to spowodowane częścią sekwencyjną naszego programu. Nawet jeśli podzieliśmy nasz problem na równe kawałki to różne węzły mogą skończyć swoje obliczenia w różnym czasie co zmusza nas na czekanie w części sekwencyjnej.



Efektywność

Efektywność utrzymuje się pomiędzy 0.8 - 1.0, co wydaje się być rozsądnym zakresem. Idealną wartością dla efektywności byłaby wartość 1. Wydaje się, że mniejsze liczby punktów osiągały lepszą efektywność. To przypuszczalnie dlatego, że różnice w czasie wykonania programu na różnych węzłach są bardziej widoczne im ten program się dłużej wykonuje, a to w efekcie zwiększa sekwencyjną część czekania na wyniki.





Część Sekwencyjna

Wykresy pokazują niezerową część sekwencyjną, która idealnie byłaby zerowa. Dla naszego problemu ma ona bardzo niską wartość.