Metody programowania równoległego

Maciej Kozieja

Część 1 - Komunikacja P2P

Programy do pomiaru danych zostały napisane w języku `python2`. Programy wykonywane są na `vnode` klastra dostępnego na `vnode-**.dydaktyka.icsr.agh.edu.pl`. Celem ćwiczenia jest zapoznanie się z wybranymi funkcjami i ich eksperymentalne wyznaczenie **przepustowości** oraz **opóźnienia**.

Wybrane funkcje to:

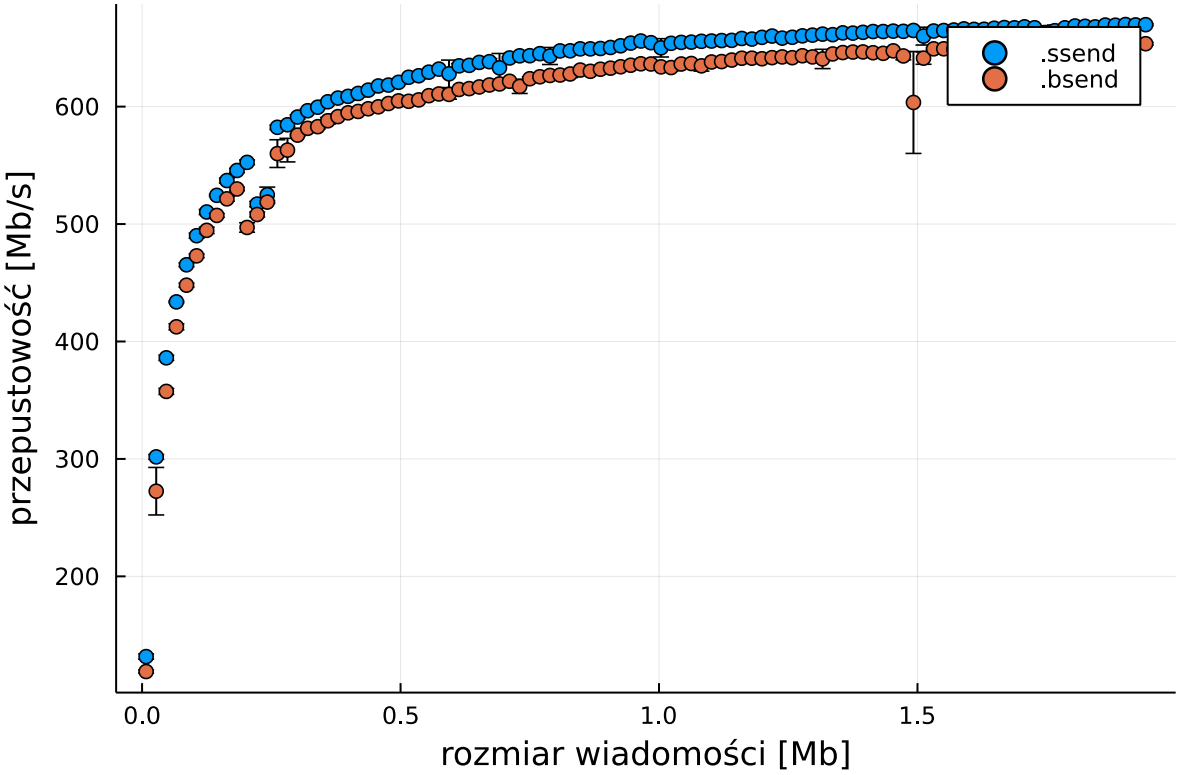
- `ssend` - Blocking send in synchronous mode.
- `bsend` - Blocking send in buffered mode.

Przepustowość

Przepustowość na pojedynczym węźle

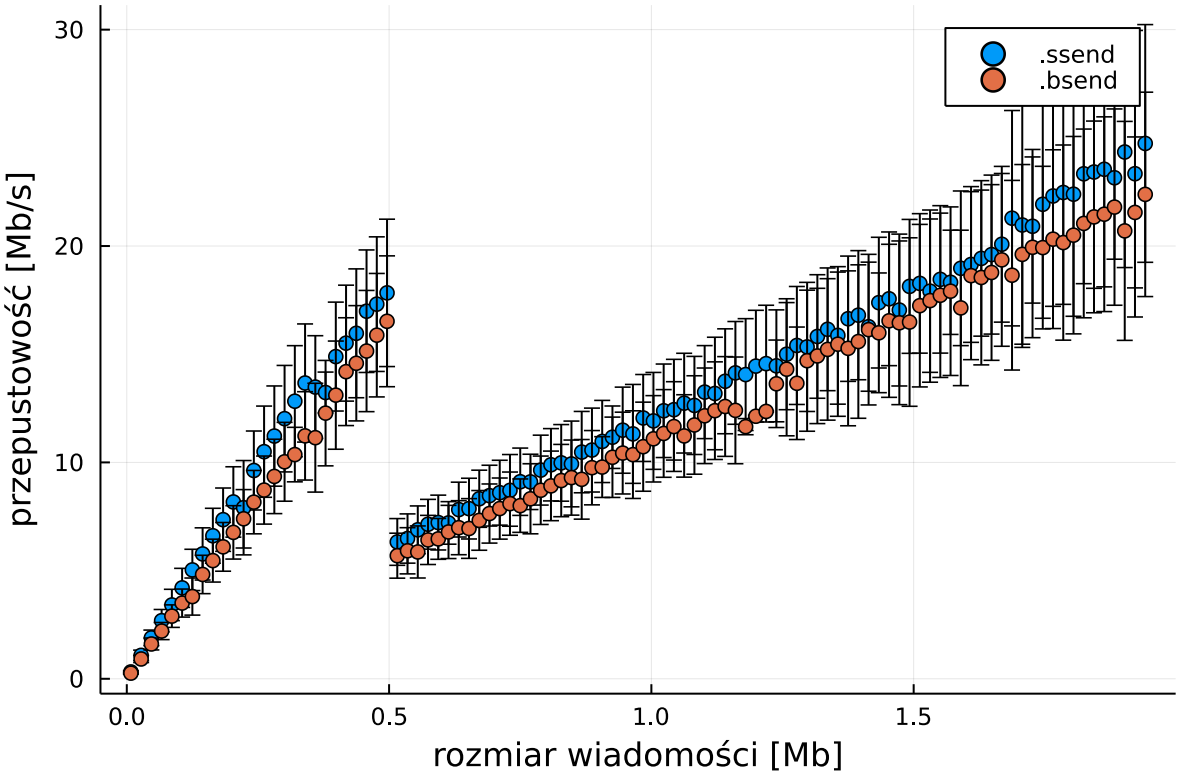
Z zebranych danych możemy zaobserwować, że wysyłanie danych z wykorzystaniem funkcji `bsend` jest wolniejsze w porównaniu do wysyłania danych z wykorzystaniem funkcji `ssend`. Jest to spowodowane dodatkowym narzutem wynikającym z kopiowania danych do bufora.

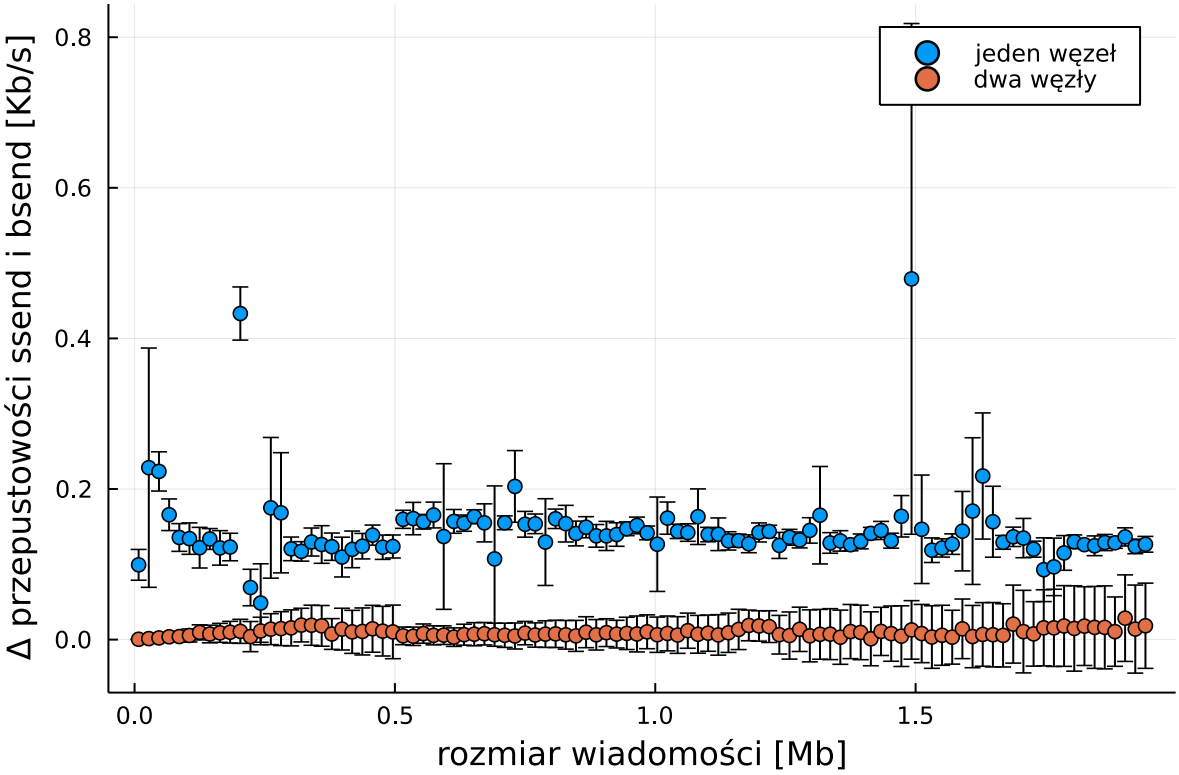
Możemy również zaobserwować, że wraz ze wzrostem rozmiaru wiadomości przepustowość zwiększa się. Wrócimy do tej obserwacji po zaobserwowaniu zachowania na dwóch węzłach.



Przepustowość na dwóch węzłach

Dane zebrane podczas pomiarów z wykorzystaniem dwóch węzłów pokazują inny trend. Odrazu rzuca się w oczy znacznie zmniejszona przepustowość. Widzimy również, że różnica pomiędzy wysyłaniem wiadomości z wykorzystaniem funkcji `ssend` i `bsend` stałą się pozornie bardziej widoczna. Jednak jeśli porównamy różnicę czasów jest ona bardzo podobna, a wręcz niższa, ponieważ narzut związany z komunikacją w tym przypadku jest większy, co zmniejsza różnicę w metodzie przesyłania.





Opóźnienie

Na podstawie zebranych danych można oszacować opóźnienie w komunikacji pomiędzy instancjami naszego programu.

W przypadku funkcji `ssend` jest to:

- jeden węzeł: 2.5e-5 ± 4.4e-5
- dwa węzły: 0.0326 ± 0.0016

W przypadku funkcji `bsend` jest to:

- jeden węzeł: 2.8e-5 ± 4.5e-5
- dwa węzły: 0.0321 ± 0.0028

Jak można zauważyć opóźnienie jest niemalże niezależne od wykorzystanej metody.

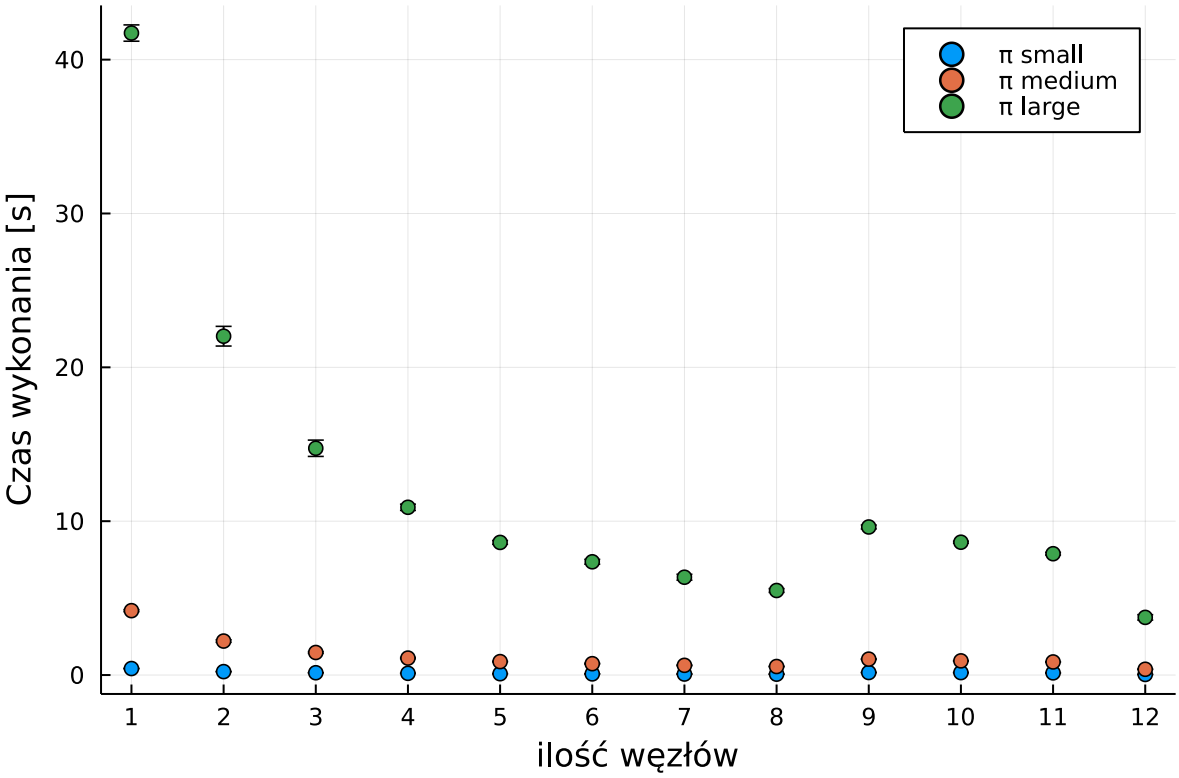
Sposób pozyskiwania danych

Programy są wykonywane kilkukrotnie w celu, ilość wysyłanych wiadomości jest stała i wynosi 200 (100 w obie strony). Wartość jest stosunkowo mała, ponieważ dokonuje wielu pomiarów i chcę zminimalizować możliwość "kolizji" z innyi osobami. Przeprowadzałem również próby z większą ilością powtorzeń, ale nie skutkowały one zwiększeniem dokładności pomiarów dlatego postanowiłem pozostać przy tej ilości.

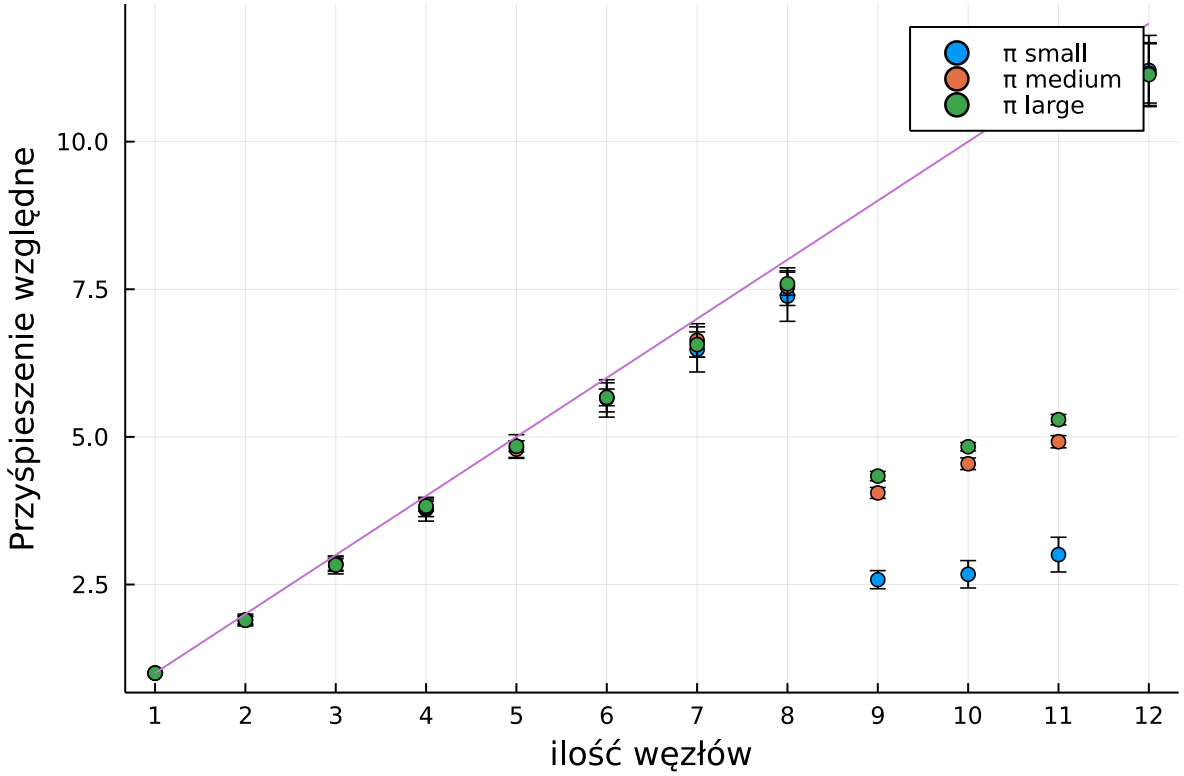
Część 2 - Badanie efektywności programu równoległego

Programy analogicznie napisany w języku `python2`. Program wykonywany jest na `prometheus`. Celem ćwiczenia jest eksperymentalne wyznaczenie wartości `przyśpieszenia`, `efektywności` oraz `części sekwencyjnej`. W tym celu stworzony został program przybliżający liczbę π wykorzystujący metodę Monte Carlo. Zaletą tego programu jest fakt, że posiada on znikomą część sekwencyjną.

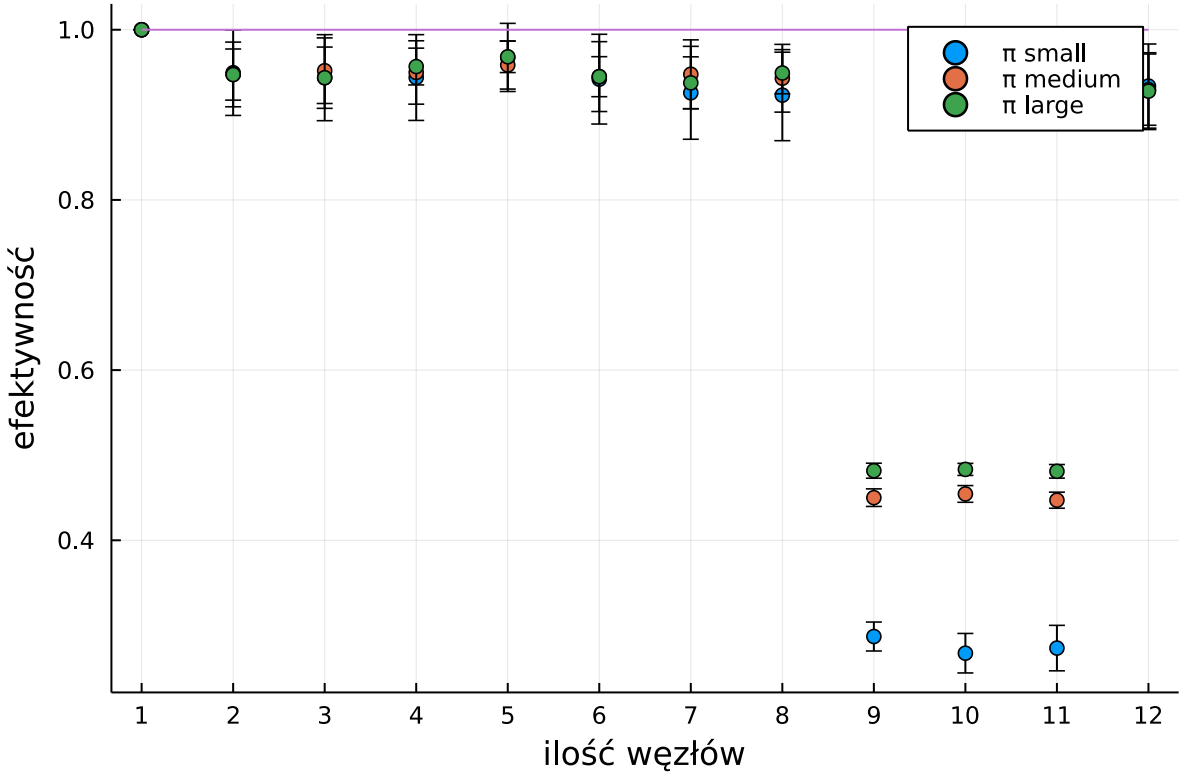
Zgodnie z oczekiwaniem czas potrzebny na wykonanie obliczenia spada wraz z liczbą procesorów proporcjonalnie do $\frac{1}{x}$.



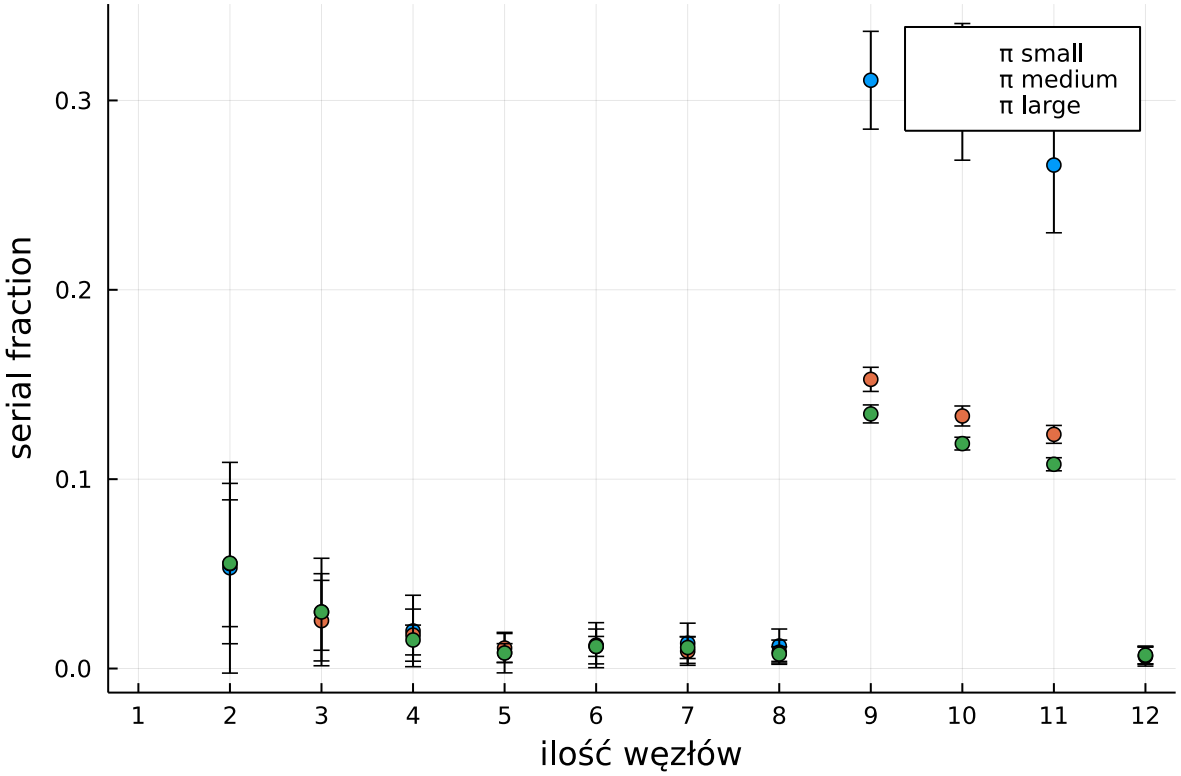
Podobnie do czasu wykonania zgodnie z oczekiwaniem, przyśpieszenie rośnie niemalże liniowo, jednak zwalnia z czasem.



Efektywność jak i zarówno przyśpieszenie są tak naprawdę zależne bezpośrednio od czasu wykonania, także również zachowują się tak jak byśmy tego oczekiwali.



Zgodnie z oczekiwaniem część sekwencyjna naszego programu powinna być bliska 0. Wydaje sie że dane potwierdzają ten trend.



Sposób pozyskiwania danych

Przygotowany przezemnie skrypt wykonywał program dla 3 określonych rozmiarów:

- π small - 4e5
- π meadium - 4e6
- π large - 4e7

Programy zostały wykonane odpowiednio z ilością węzłów od 1 do 12, a całość została powtórzona 20 razy.

```
toMb (generic function with 1 method)

measure (generic function with 1 method)

measure2size (generic function with 1 method)

plot_sf! (generic function with 1 method)

plot_speedup! (generic function with 1 method)

plot_efficiency! (generic function with 1 method)

plot_time! (generic function with 1 method)

delay (generic function with 1 method)
```