Badania porównawcze dwóch rozwiązań PK z losowością - bez zagłodzenia.

Dominik Szot Gr. 12

Grupa badawcza:

- Dominik Szot
- Bartłomiej Słupik
- Hubert Kabziński
- Aleksandra Poskróbek

Badania realizowaliśmy na wspólnym kodzie.

Opis implementacji obu porównywanych rozwiązań

Rozwiązanie problemu producent konsument na 1 locku i 4 conditions

- ### Na rozwiązanie składają się:

    - **Lock :** Blokada jest używana do zabezpieczenia dostępu do bufora, który jest współdzielony między producentami a konsumentami.

    - **Bufor:** Wspólny obszar pamięci służący do komunikacji między producentami i konsumentami udostępniający dwie procedury: wstawiania i pobierania.

    - **Conditions:** Zmienne/warunki na których procesy są wstrzymywane czekając aż bufer osiągnie wymagany stan.
        - PIERWSZYKONS: warunek na którym czeka obecnie pierwszy do obsłużenia konsument
        - PIERWSZYPROD: warunek na którym czeka obecnie pierwszy do obsłużenia producent
        - RESZTAKONS: warunek na którym czekają konsumenci, którzy napotkali na zajętego firstConst.
        - RESZTA PROD: warunek na którym czekają konsumenci, którzy napotkali na zajętego firstProdicerCondition.

- ## Rozwiązanie

  Zarówno producent jak i konsument sprawdza, czy nikt nie czeka przed nim w kolejce (warunek PIERWSZYKONS lub PIERWSZYPROD). Jeśli nikt nie czeka, proces staje się pierwszym (wszystkie inne procesy stają się RESZTĄ) a jeśli bufer posiada zasoby - proces modyfikuje bufer. W przypadku gdy bufer nie posiada zasobów proces czeka. Po zakończeniu proces wysyła sygnał najpierw reszcie procesów tego samego typu (RESZTAPROD / RESZTAKONS), następnie pierwszemu procesowi przeciwnego typu (PIERWSZYKONS / PIERWSZYPROD).

Rozwiązanie problemu producent - konsument na 3 lockach i 1 condition

- ## Na rozwiązanie składają się:

  - **Bufor:** Wspólny obszar pamięci służący do komunikacji między producentami i konsumentami udostępniający dwie procedury: wstawiania i pobierania.

  - **Lock:** używany do zabezpieczenia dostępu do bufora,
    - commonLock: główna blokada zabezpieczania dostępu do bufora
    - consumerLock: pozwala wejść tylko jednemu konsumentowi
    - producerLock: pozwala wejść tylko jednemu producentowi

  - **Condition:** (commonCondition) warunek na którym czekają procesy króre nie spełniają warunków buforu (za mało miejsca / za duża porcja do pobrania)

- ## Rozwiązanie

  Zarówno producent jak i konsument starają się najpierw uzyskać odpowiedni typ Lock'a (consumerLock / producerLock), a następnie stajają się uzyskać commonLock. Gdy proces spełni te założenia, jeśli może wykonać zadanie, wykonuje je i wysyła sygnał do wszystkich wątków czekających na commonCondition. Następnie proces zwalnia Locki w odwrotnej kolejności. W przypadku jeśli producent/konsument nie może wykonać zadania – wchodzi na warunek commonCondition gdzie czeka aż bufor będzie posiadał niezbędne zasoby.

Sposób przeprowadzenia eksperymentu

Dla stałej liczby wątków, stałej liczby pętli na wątek oraz stałej długości bufora zmierzyliśmy czas rzeczywisty i czas procesora uśredniając 50 pomiarów dla kombinacji:

- [Typ bufora]: 4-Conditions / 3-Lock
- Maksymalny rozmiar porcji w zakresie od 50 do 500.

Liczbę pętli ustaliliśmy na 10000 iteracji, rozmiar bufora na 1000 jednostek, natomiast liczbę wątków w kolejnych badaniach na :

- 12 wątków w pierwszym eksperymencie (6 producentów 6 konsumentów)
- 20 wątków w drugim eksperymencie (10 producentów 10 konsumentów)
- 60 wątków w trzecim eksperymencie (30 producentów 30 konsumentów)

Kolejne kroki eksperymentu:

- Stworzenie wszystkich wątków
- Rozpoczęcie pomiaru czasu
- Uruchomienie wszystkich wątków
- Sprawdzanie w pętli, czy któryś z wątków zakończył działanie
- Pomiar czasu rzeczywistego
- Zebranie czasów procesora ze wszystkich wątków i zsumowanie ich
- Zabicie wątków, które jeszcze działają

Podczas pomiaru czasu CPU używając ThreadMXBean, czas wątku zakończonego został uśredniony do czasu najdłużej działającego wątku. Jest to spowodowane faktem, że rozwiązania ThreadMXBean nie dają możliwości sprawdzenia czasu CPU dla zakończonego wątku.

Urządzenia

Eksperyment przeprowadziliśmy na następujących urządzeniach:

- **Urządzenie 1 :**
  Procesor: AMD Ryzen™ 5 5600U
  (6 rdzeni, 12 wątków, 2.30–4.20 GHz, 19 MB cache)
  Ram: 16.0 GB
  System: Ubuntu 22.04.3
  program uruchamiany przez IntelliJ (Java 17)

- **Urządzenie 2:**
  Procesor: Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz 2.59 GHz
  8 rdzeni, 16 wątków
  Ram: 16.0 GB
  System: Windows 10 Home x64
  program uruchamiany przez IntelliJ (Java 17)

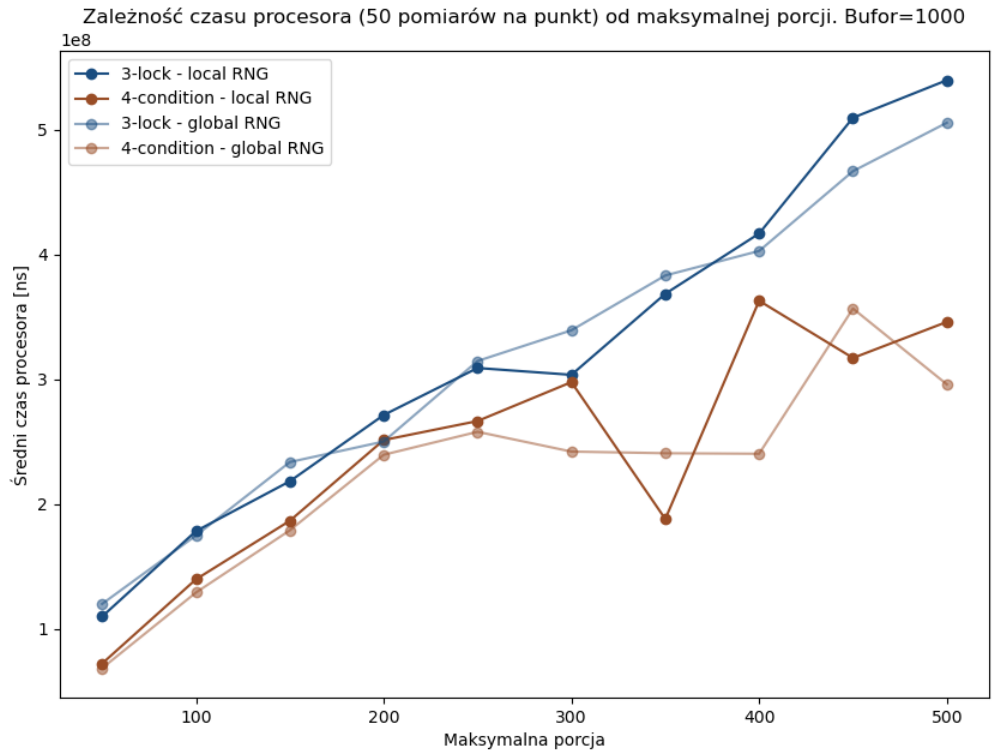
- **Urządzenie 3 :**

  Procesor: AMD Ryzen 7 4800H, 8 rdzeni, 16 wątków, 2.9 GHz
  Ram: 32 GB
  System: Windows 10 Home
  Program uruchamiany przez IntelliJ (Java 17)

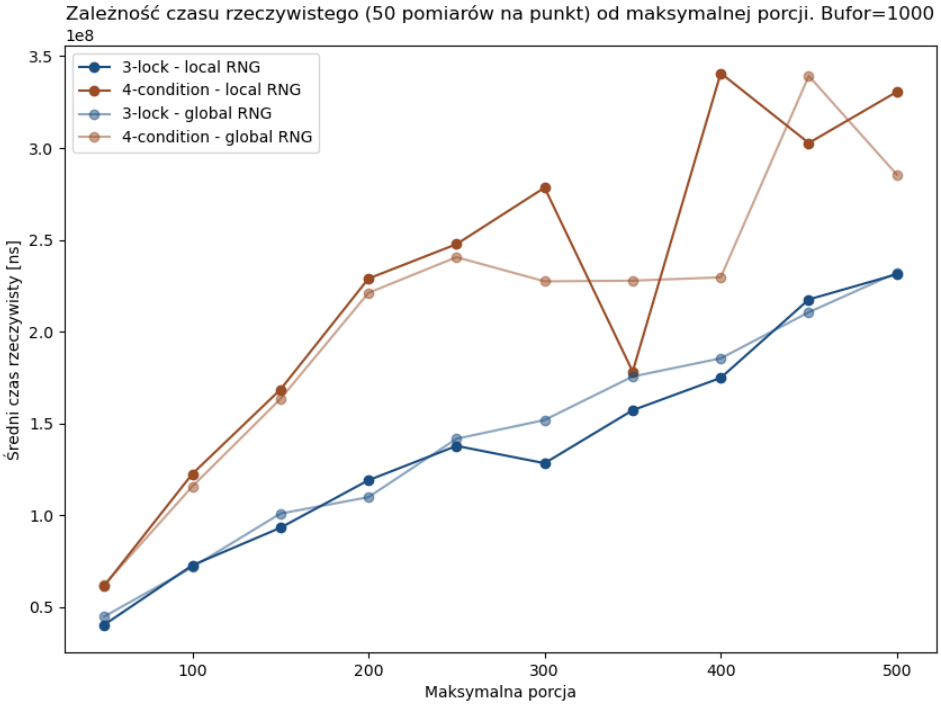
- **Urządzenie 4 :**

  Procesor: Intel(R) Core(TM) i5-7500 CPU (4 rdzenie, 4 wątki @ 3.40GHz 3.40 GHz)
  Ram: 16.0 GB
  System: Windows 10 Home x64

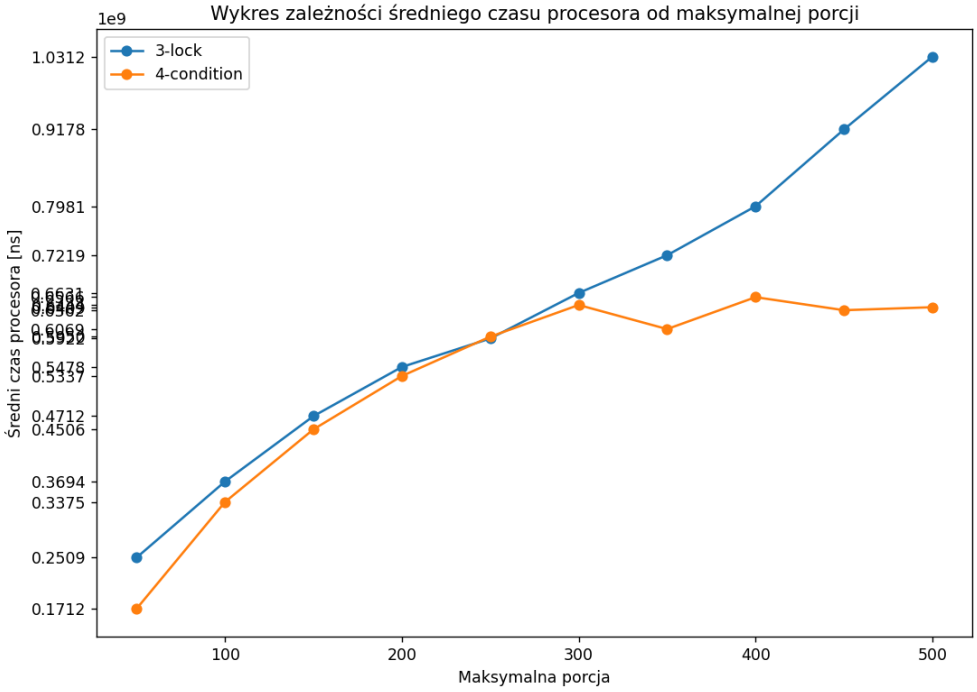
Pomiary na różnych urządzeniach dla 6 producentów i 6 konsumentów



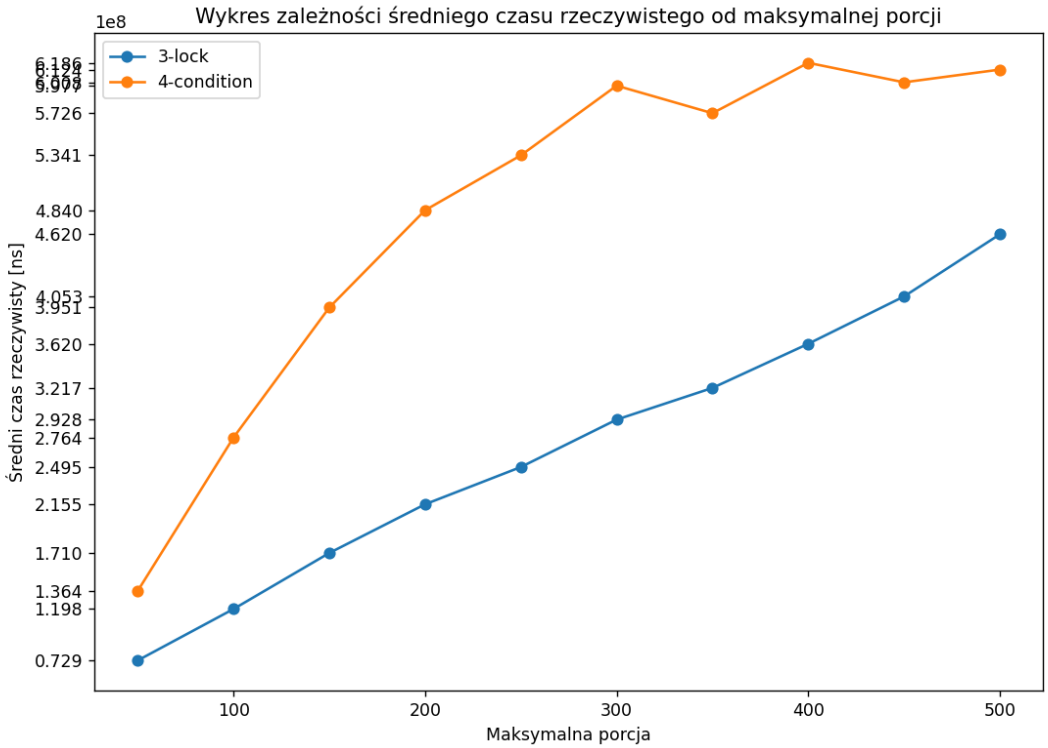
Rysunek 1: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 1 korzystając z 12 wątków.



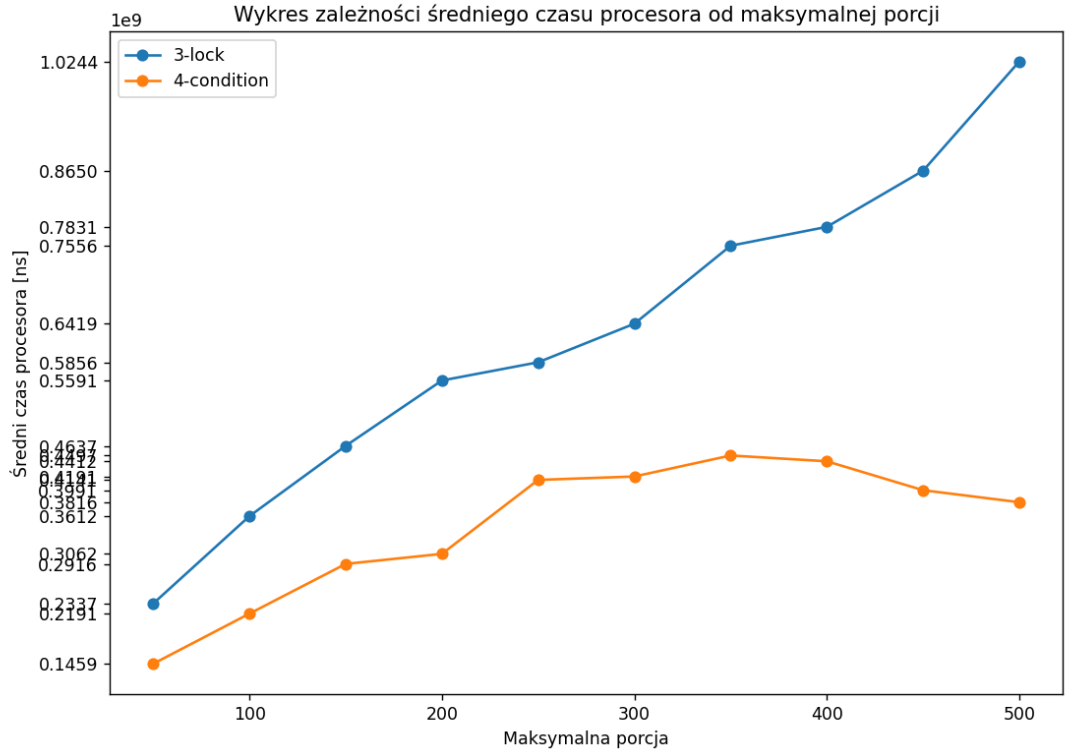
Rysunek 2: Zależność średniego czasu rzeczywistego od maksymalnej porcji dla urządzenia nr 1 korzystając z 12 wątków.



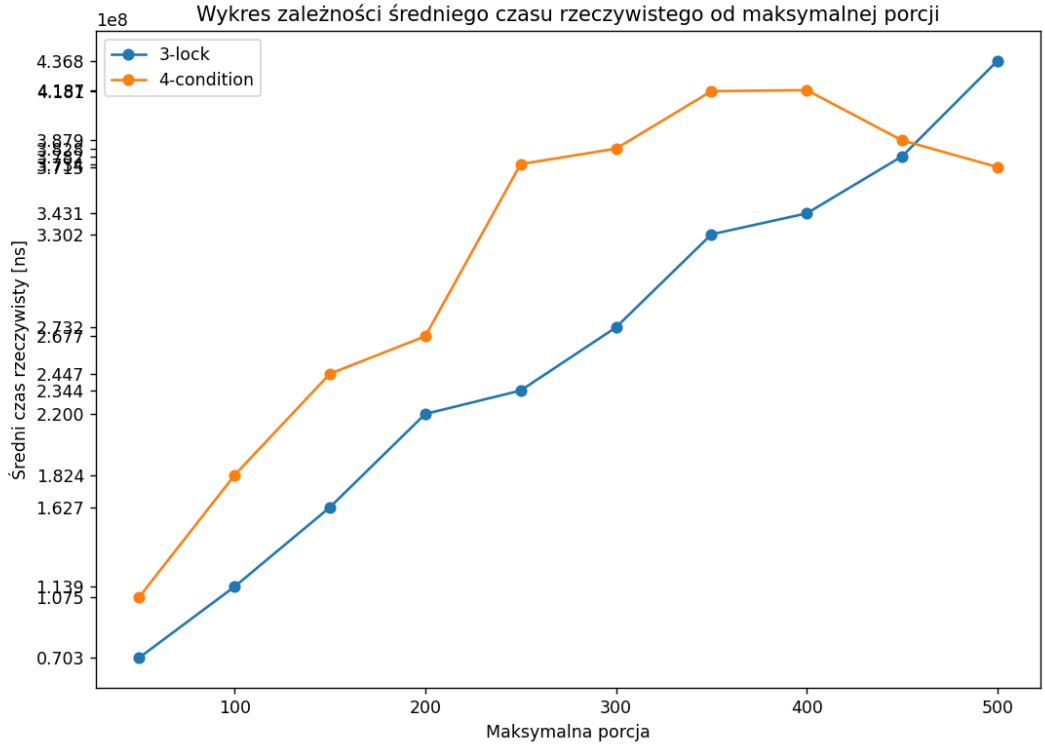
Rysunek 3: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 2 korzystając z 12 wątków oraz Globalnego generatora porcji.



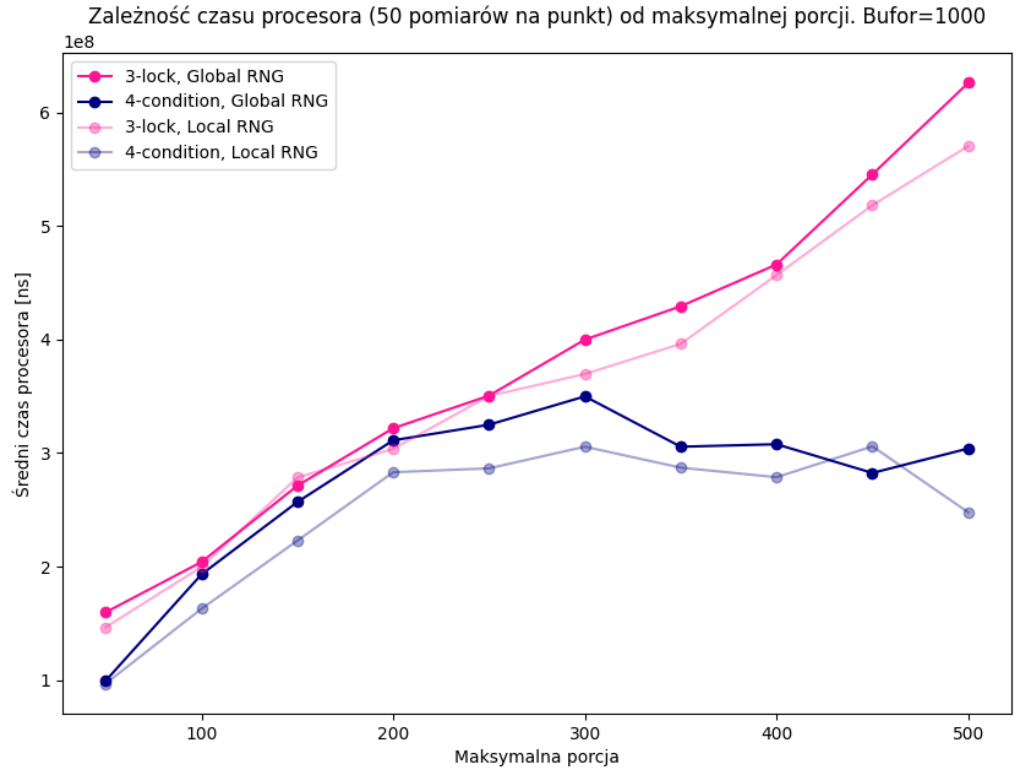
Rysunek 4: Zależność średniego czasu rzeczywistego od maksymalnej porcji dla urządzenia nr 2 korzystając z 12 wątków oraz Globalnego generatora porcji.



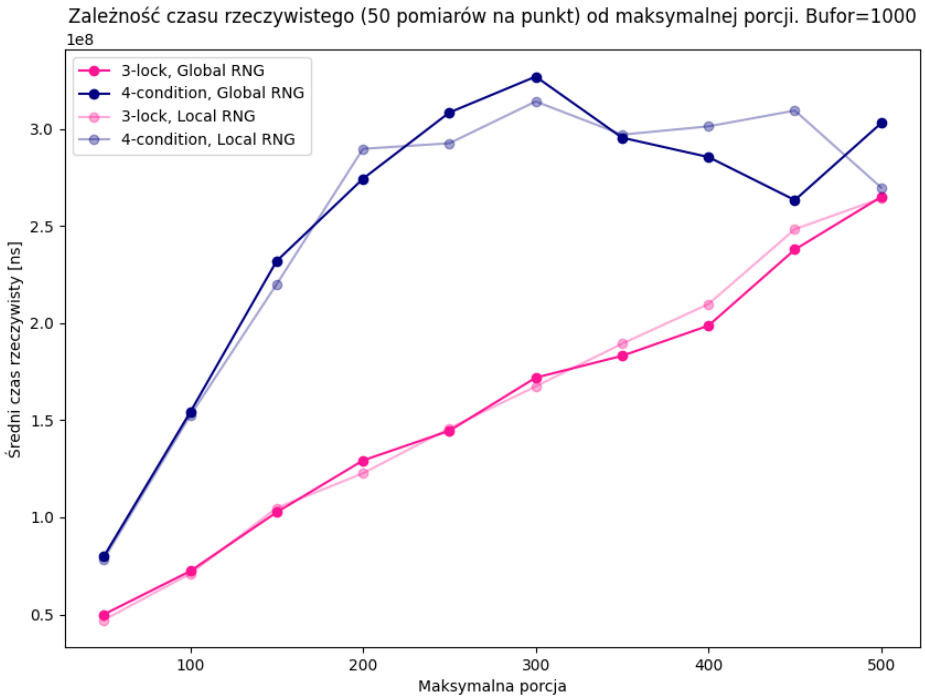
Rysunek 5: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 2 korzystając z 12 wątków oraz Lokalnego generatora.



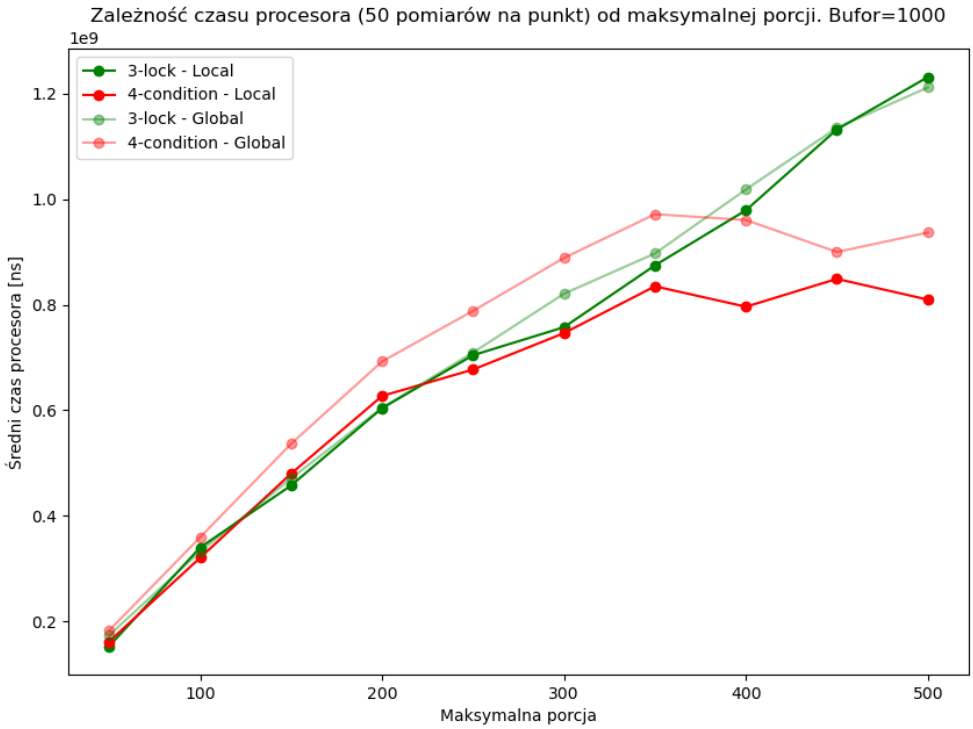
Rysunek 6: Zależność średniego czasu rzeczywistego od maksymalnej porcji dla urządzenia nr 2 korzystając z 12 wątków oraz Lokalnego generatora porcji.



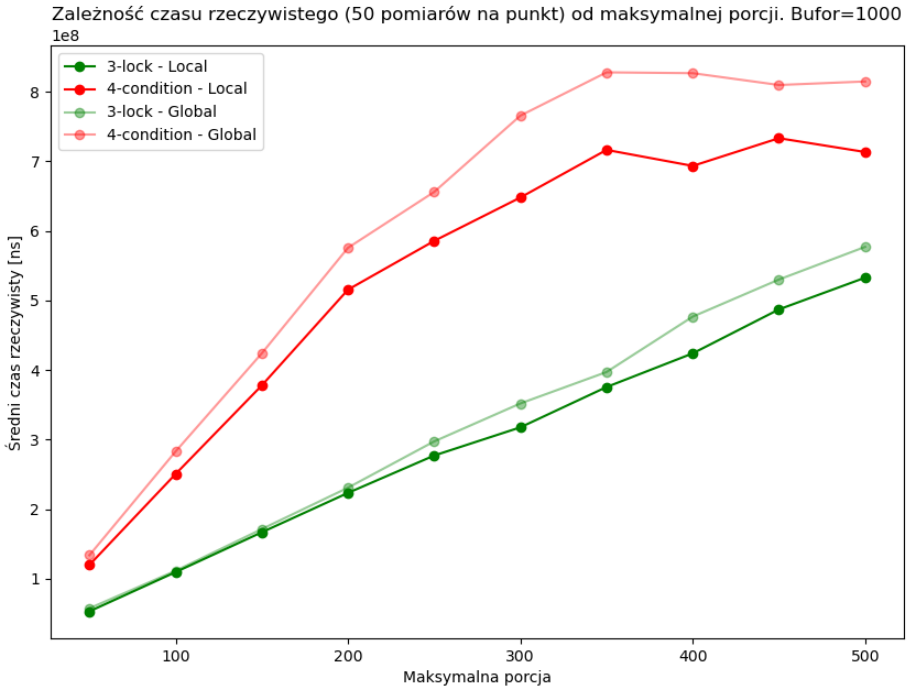
Rysunek 7: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 3 korzystając z 12 wątków.



Rysunek 8: Zależność średniego czasu rzeczywistego od maksymalnej porcji dla urządzenia nr 1 korzystając z 12 wątków.

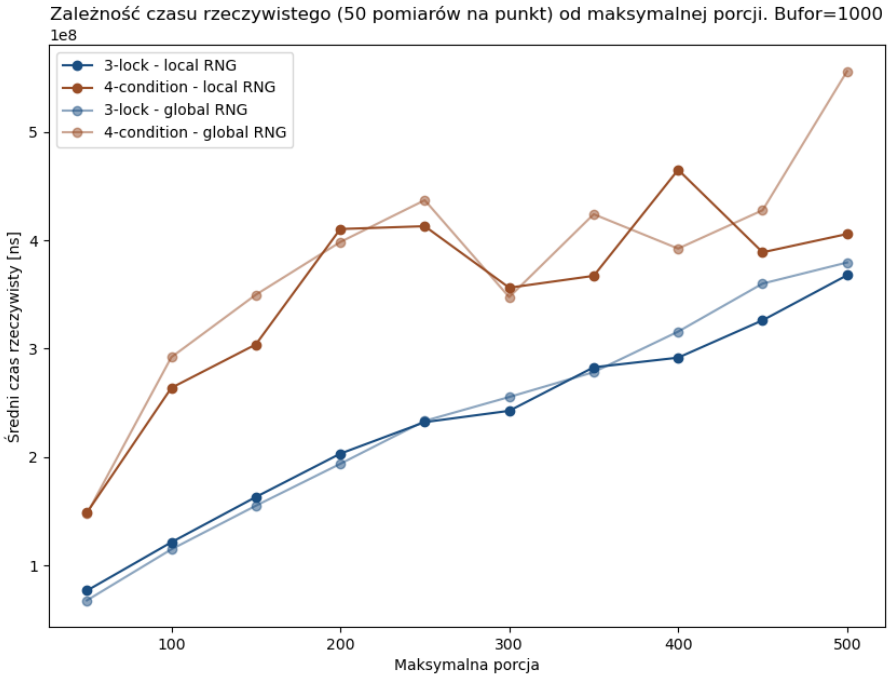


Rysunek 9: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 4 korzystając z 12 wątków.

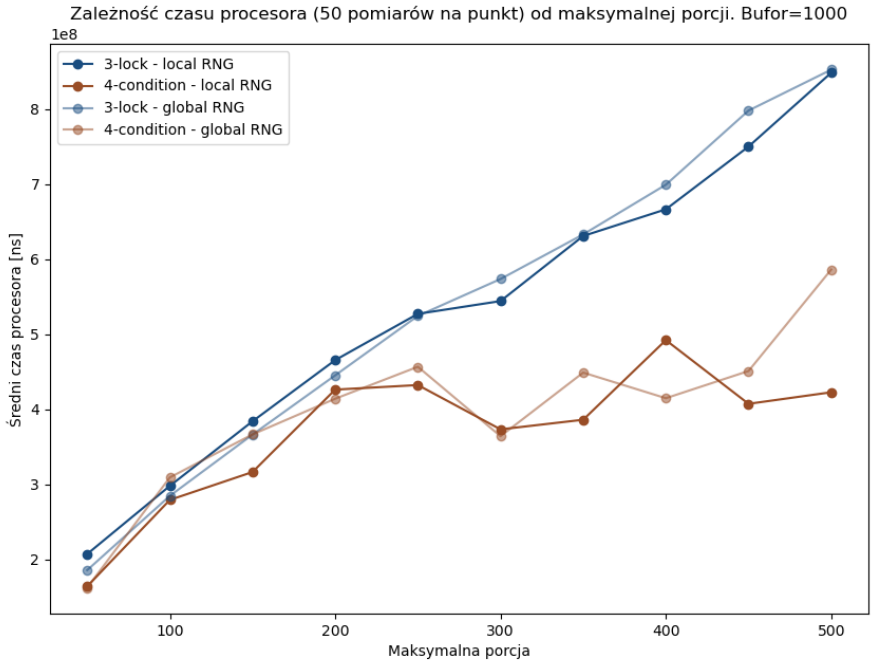


Rysunek 10: Zależność średniego czasu rzeczywistego od maksymalnej porcji dla urządzenia nr 1 korzystając z 12 wątków.

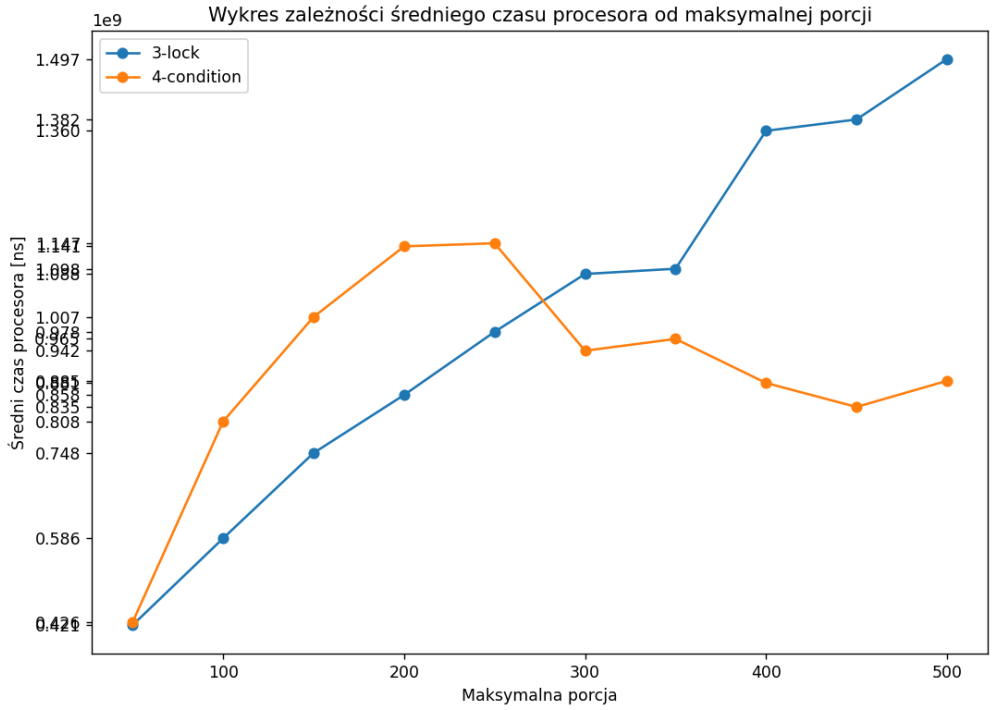
Pomiary na różnych urządzeniach dla 10 producentów i 10 konsumentów



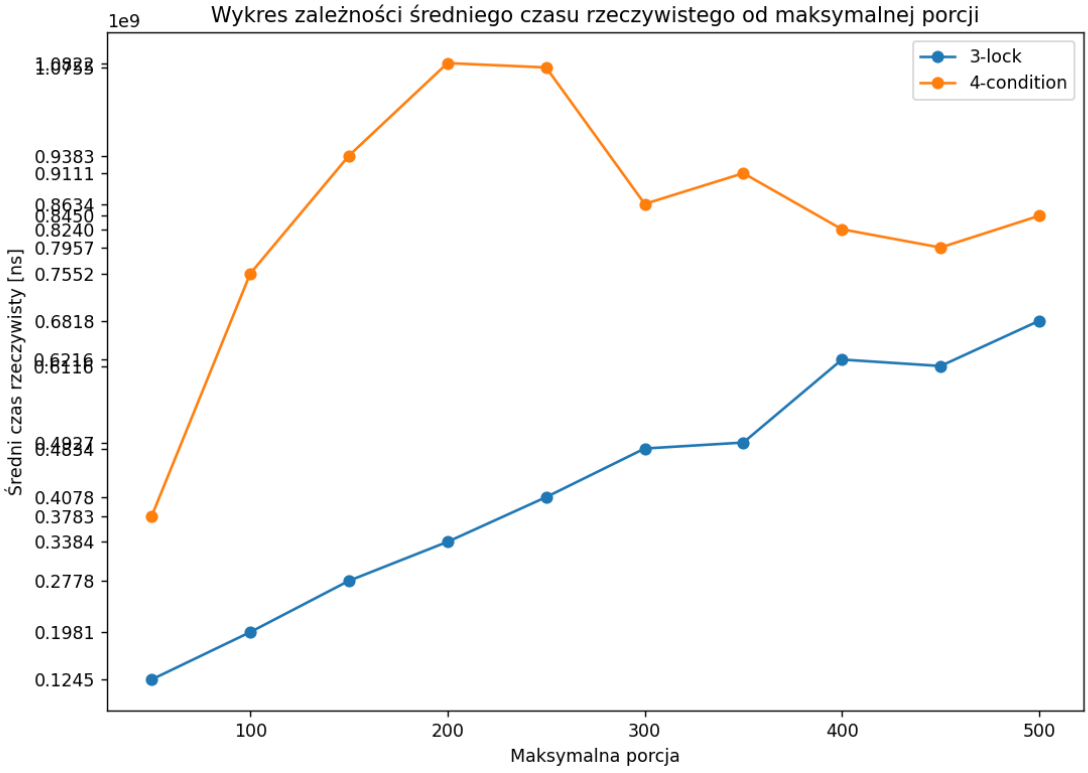
Rysunek 11: Zależność średniego czasu rzeczywistego od maksymalnej porcji dla urządzenia nr 1 korzystając z 20 wątków.



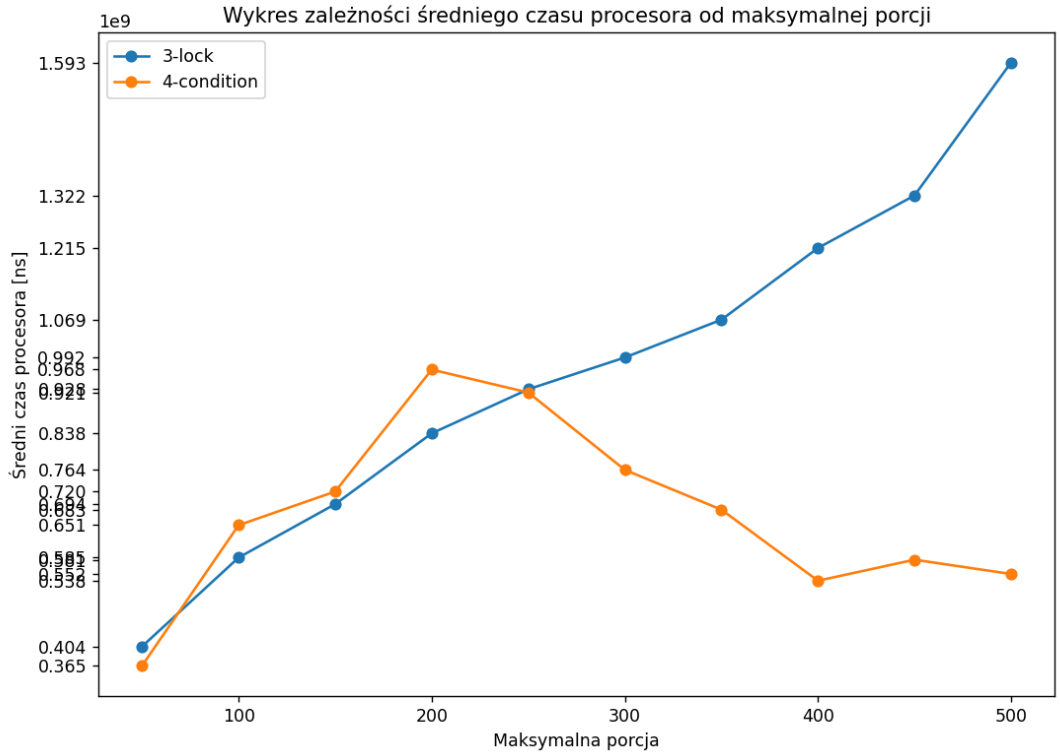
Rysunek 12: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 1 korzystając z 20 wątków.



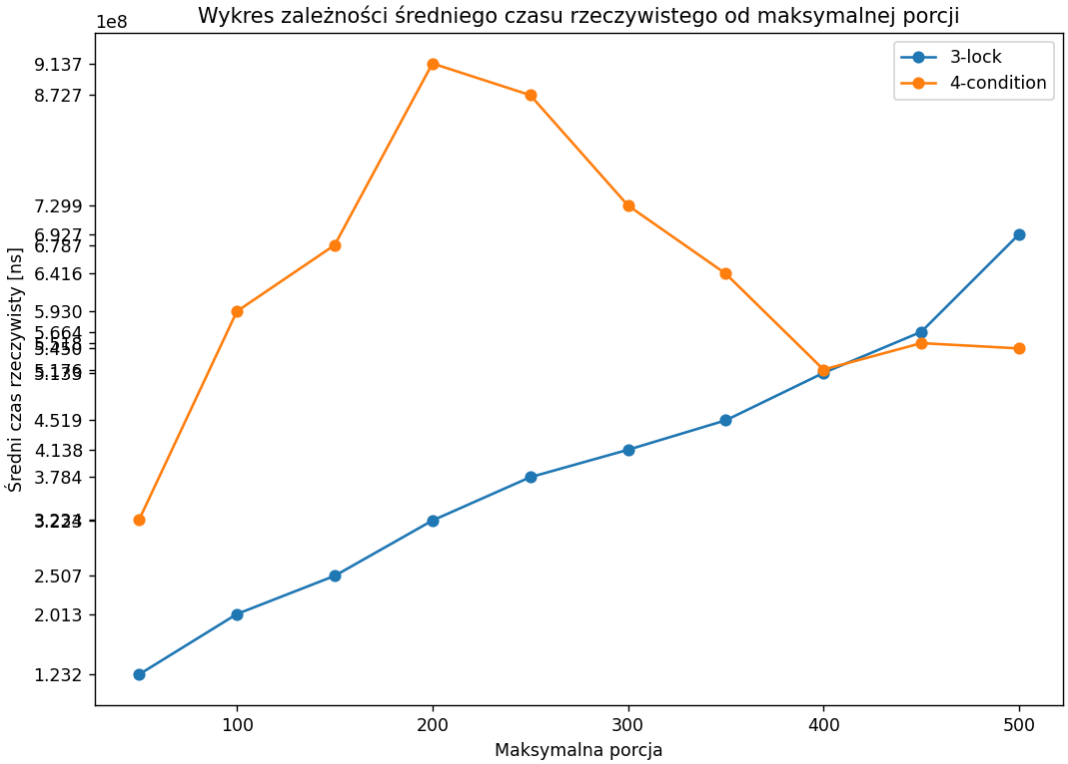
Rysunek 13: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 2 korzystając z 20 wątków oraz Globalnego generatora porcji.



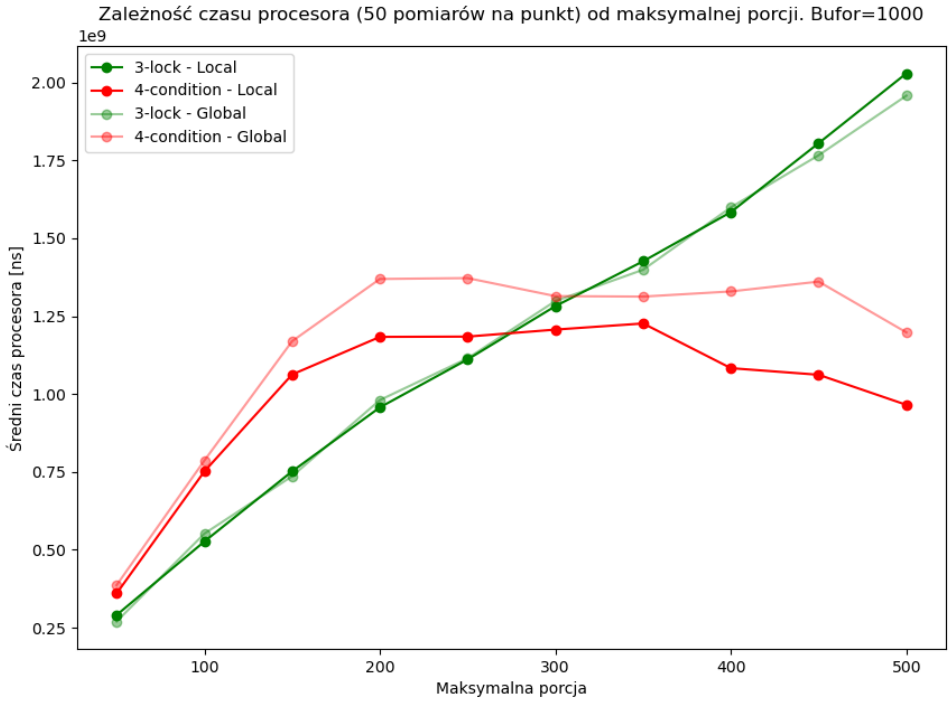
Rysunek 14: Zależność średniego czasu rzeczywistego od maksymalnej porcji dla urządzenia nr 2 korzystając z 20 wątków oraz Globalnego generatora porcji.



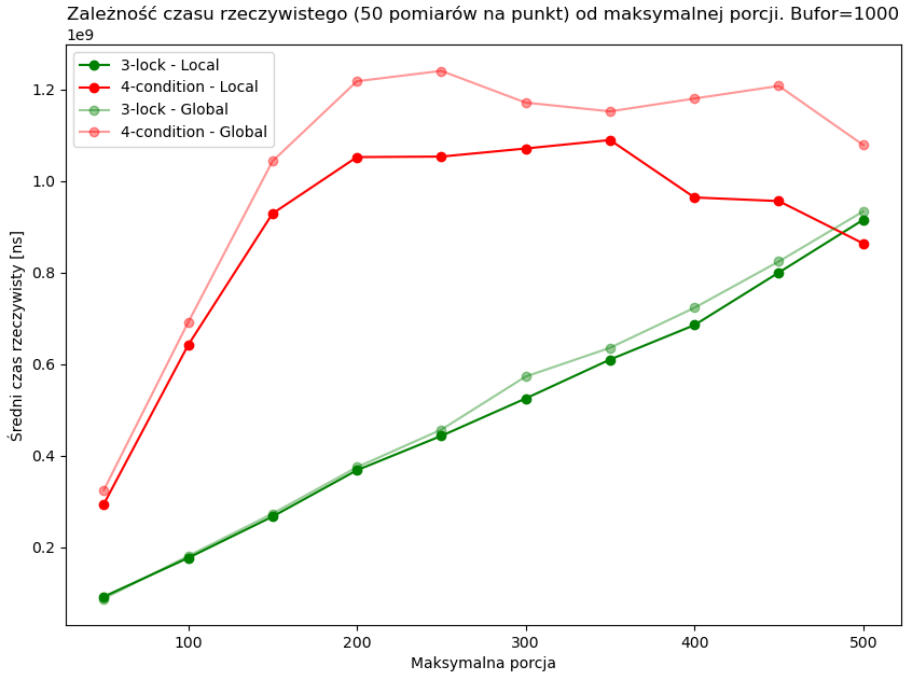
Rysunek 15: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 2 korzystając z 20 wątków oraz Lokalnego generatora porcji.



Rysunek 16: Zależność średniego czasu rzeczywistego od maksymalnej porcji dla urządzenia nr 2 korzystając z 20 wątków oraz Lokalnego generatora porcji.

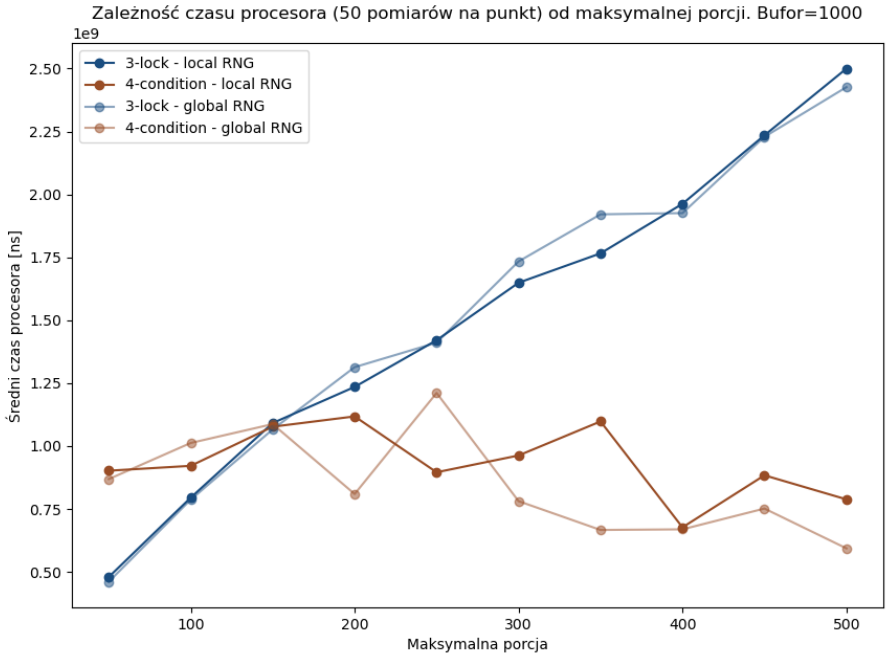


Rysunek 17: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 4 korzystając z 20 wątków.

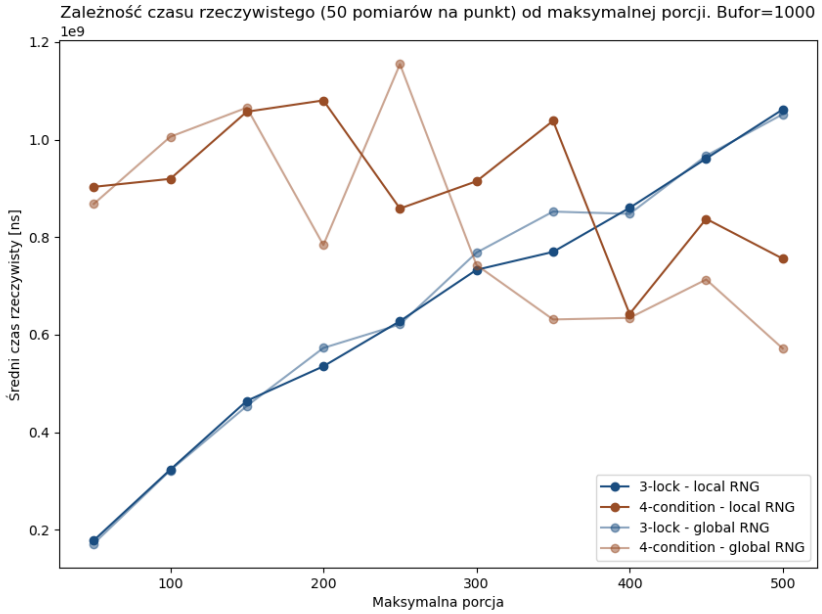


Rysunek 18: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 4 korzystając z 20 wątków.

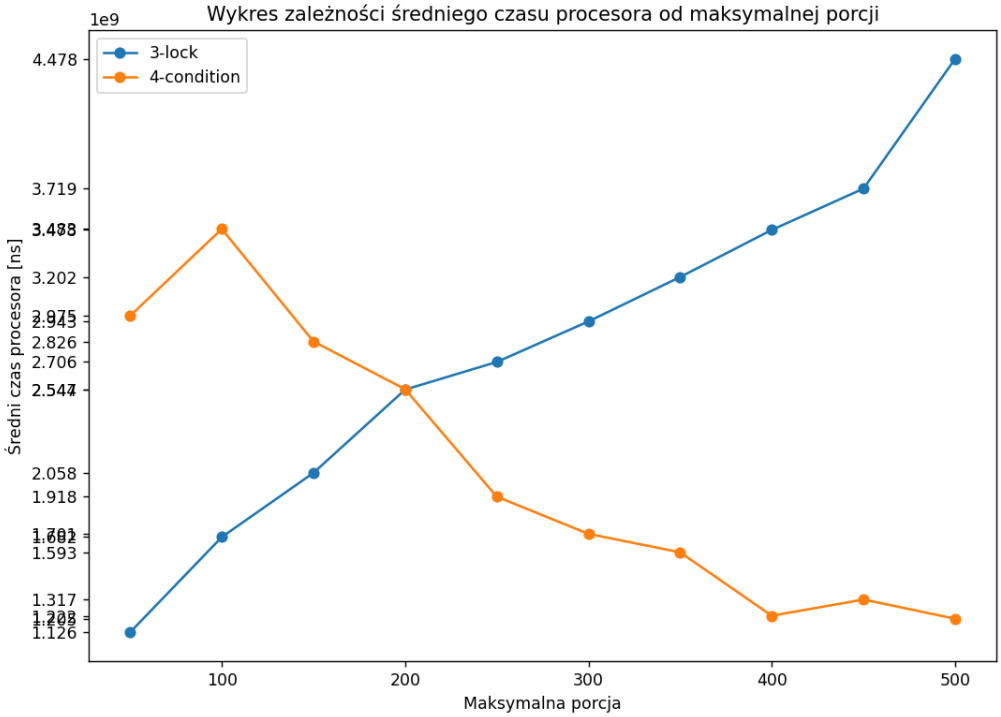
Pomiary na różnych urządzeniach dla 30 producentów i 30 konsumentów



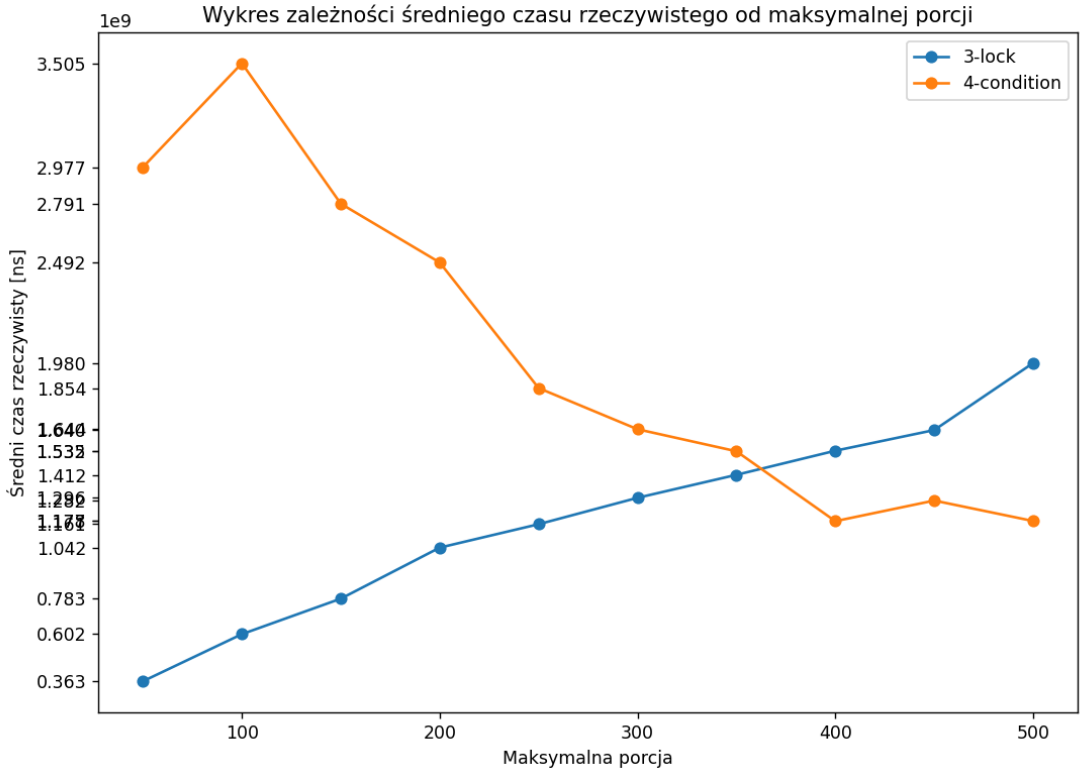
Rysunek 19: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 1 korzystając z 30 wątków.



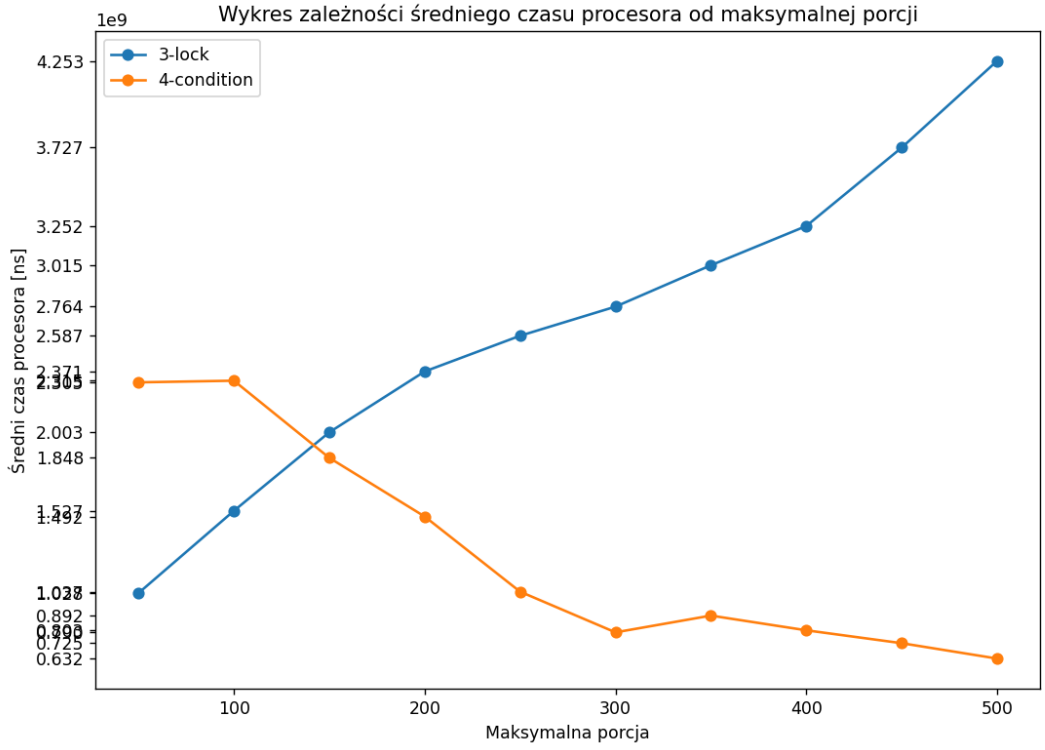
Rysunek 20: Zależność średniego czasu rzeczywistego od maksymalnej porcji dla urządzenia nr 1 korzystając z 30 wątków.



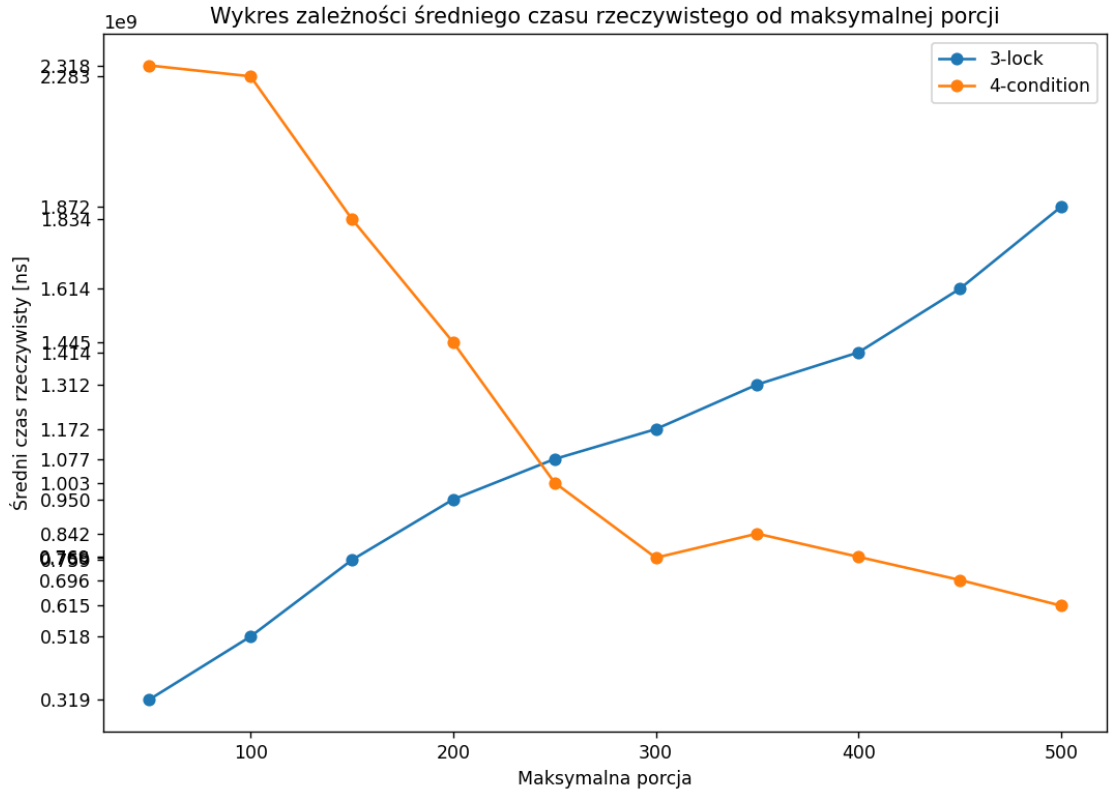
Rysunek 21: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 2 korzystając z 30 wątków oraz Globalnego generatora porcji.



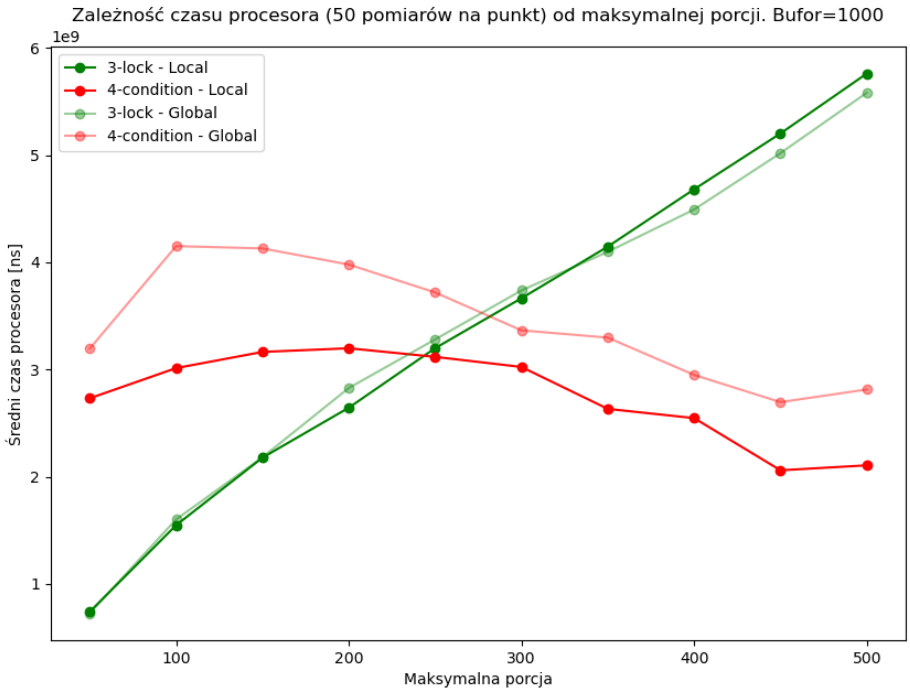
Rysunek 22: Zależność średniego czasu rzeczywistego od maksymalnej porcji dla urządzenia nr 2 korzystając z 30 wątków oraz Globalnego generatora porcji.



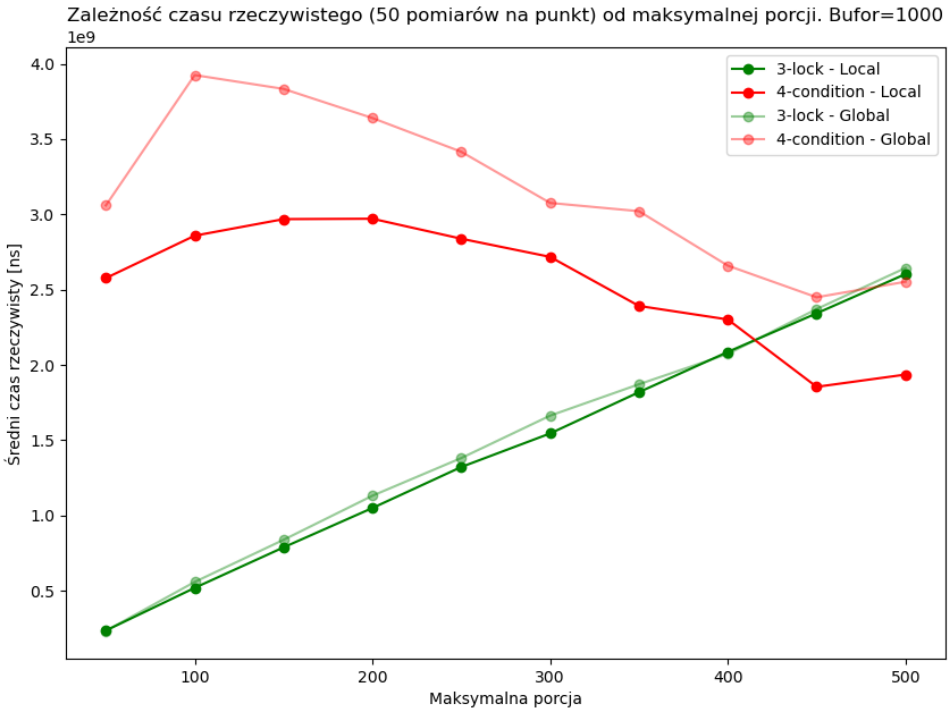
Rysunek 23: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 2 korzystając z 30 wątków oraz Lokalnego generatora porcji.



Rysunek 24: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 2 korzystając z 30 wątków oraz Lokalnego generatora porcji.



Rysunek 25: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 4 korzystając z 30 wątków.



Rysunek 26: Zależność średniego czasu procesora od maksymalnej porcji dla urządzenia nr 4 korzystając z 30 wątków.

Wnioski

Wybór optymalnego rozwiązania problemu, według naszego eksperymentu zależeć powinien od maksymalnej losowanej porcji oraz ilości wątków. Rozwiązanie oparte na 4-Conditions radziło sobie, lepiej od rozwiązania na 3-Lockach dla małej ilości porcji. Dla większych porcji (o rozmiarze połowy bufora), rozwiązanie na 4-Conditions radziło sobie zaskakująco dobrze, często deklasując rozwiązanie na 3-Lockach.

W wynikach najbardziej zastanawiają czasy rozwiązania na 3-Lockach.
Z nieznanego powodu istnieje spora różnica pomiędzy czasem działania programu liczonego względem czasu procesora a czasu rzeczywistego. Różnica ta jest proporcjonalnie większa niż w przypadku rozwiązania konkurencyjnego.

Negatywny wpływ urządzenia oraz systemu należało by wykluczyć, ponieważ wyniki pomiarów dla wszystkich urządzeń wyglądały bardzo podobnie.

Negatywny wpływ na wynik mógł mieć sposób kończenia pracy wątków, lecz sprawdzenie tego wymagałoby dalszych eksperymentów.

Nasz eksperyment nie wykazał jednoznacznie która metoda jest lepsza. W tym celu należałoby przeprowadzić ponownie eksperyment, stosując zmienioną metodę badawczą, na przykład zbadać ilość wykonanych operacji w danej jednostce czasu.