Raport: zadanie zaliczeniowe z MPI

Wojciech Żółtak (292583)
11 maja 2013

1 WSTĘP

Niniejszy dokument zawiera opis rozwiązania zadania zaliczeniowego z MPI oraz jego ewaluacji na klastrze obliczeniowym Halo2 w ICM.

2 ROZWIĄZANIE

2.1 CECHY PROBLEMU

Celem ćwiczenia było zrównoleglenie gotowego programu wykonującego wiele iteracji obliczeń w sekwencyjny sposób, z zachowaniem użytego algorytmu numerycznego.

Posiada on następujące cechy:

- Obliczenia odbywają się na kwadratowej siatce NxN.
- Każda iteracja ma dwie fazy.
- Każda faza polega na wyliczeniu nowej wartości dla połowy punktów z siatki.
- Wartość w każdym punkcie zależy tylko od wartości punktów sąsiednich.
- Wykonanie kolejnej iteracji zależy od największej różnicy między starą a nową wartością w punktach siatki.

Wynika z nich, że:

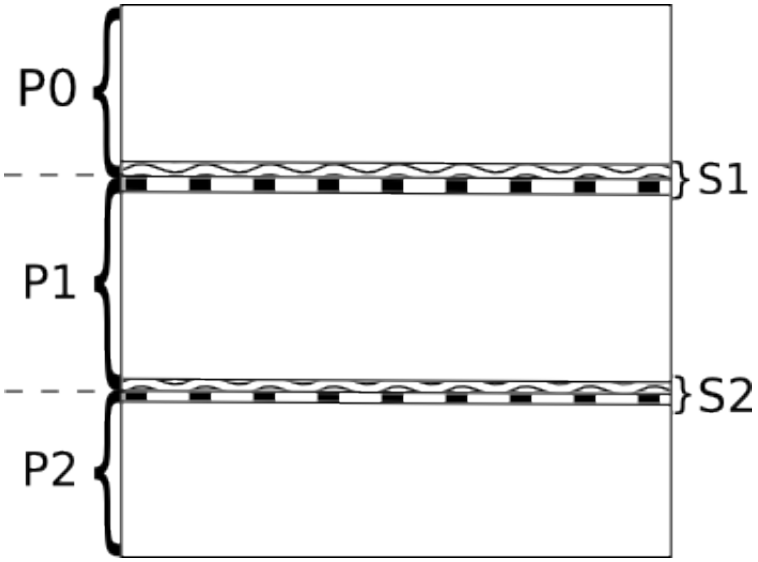
- Rozproszenie może polegać właściwie tylko na podziale obszaru siatki między węzły robocze.
- Synchronizacja węzłów musi następować co najmniej przed każdą z faz iteracji oraz po jej wykonaniu.

2.2 Opis rozwiązania

Rozwiązanie opiera się na podziale obszaru roboczego na pasy szerokości $\lceil N/P \rceil$, gdzie N to długość boku siatki, a P ilość procesów biorących udział w obliczeniu. Pasy zostają przydzielone po jednym na proces, począwszy od procesu z rangą 0. Procesy, ktorym nie starczy pasów nie pracują.

Dane generowane są przez proces o randze 0, a następnie rozsyłane do odpowiednich węzłów w wiadomościach nie większych niż 1GB. Przesyłanie większych wiadomości powodowało błędy infrastruktury Infiniband używanej w klastrze.

Ponieważ obliczenia wymagają wartości z sąsiednich punktów, zachodzi potrzeba synchronizacji części danych pomiędzy procesami przed każdą z faz iteracji. Wystarczy jednak przesłać jedynie dolną/górną krawędź pasa do „sąsiednich" procesów. Odbywa się to przy pomocy asynchronicznej komendy „`MPI_Isend`".



Rysunek 2.1: Podział obszaru roboczego między trzy procesy P1, P2 i P3 wraz z dwoma synchronizowanymi obszarami S1 i S2 na krawędziach sąsiednich pasów.

Po zakończeniu iteracji procesy dokonują redukcji obliczonych różnic wartości za pomocą funkcji minimum, a następnie sprawdzają czy należy kontynuować obliczenia.

Wynik obliczeń jest agregowany i wypisywany przez proces o randze zero, zbierający dane od pozostałych węzłów.

3 Ewaluacja

3.1 Wykonane testy

Parametry wykonanych testów ilustruje poniższa tabela:

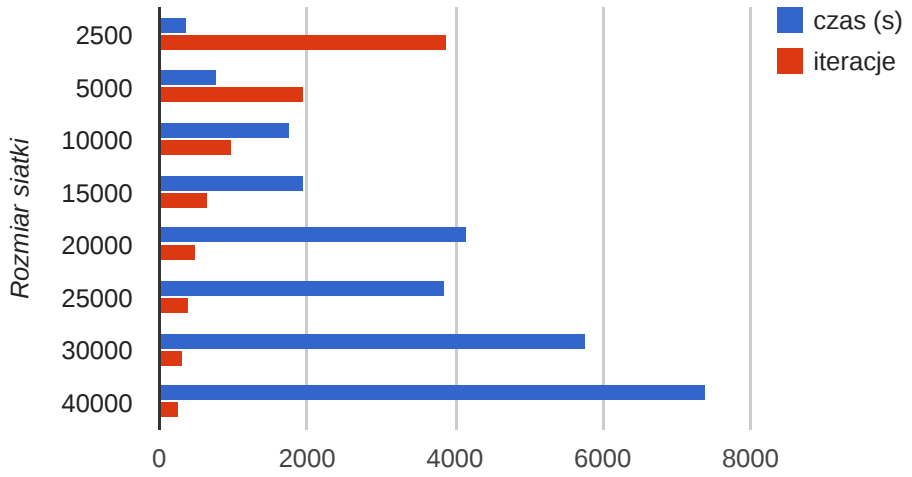
Z powodu dużego obciążenia klastra wszystkie poza jednym (`laplace-par-10000-4-2`; użyty do sprawdzenia wariancji czasów) zostały wykonane jednokrotnie.

	`laplace-seq`	`laplace-par`
Rozmiar boku siatki	2500, 5000, 10000, 15000, 20000, 25000, 30000, 40000	
Zasoby	1-1	1-1, 1-2, 2-1, 1-4, 4-1, 1-8, 8-1, 16-1, 1-16, 2-2, 2-4, 4-2, 4-8, 8-4, 8-16, 16-8

Rysunek 3.1: Rozmiary siatki oraz przydzielone zasoby dla programów podczas testów. Notacja „X-Y” oznacza X maszyn po Y rdzeni.

3.2 Program sekwencyjny

Poniższy wykres ukazuje czas działania programu sekwencyjnego dla różnych rozmiarów siatki:

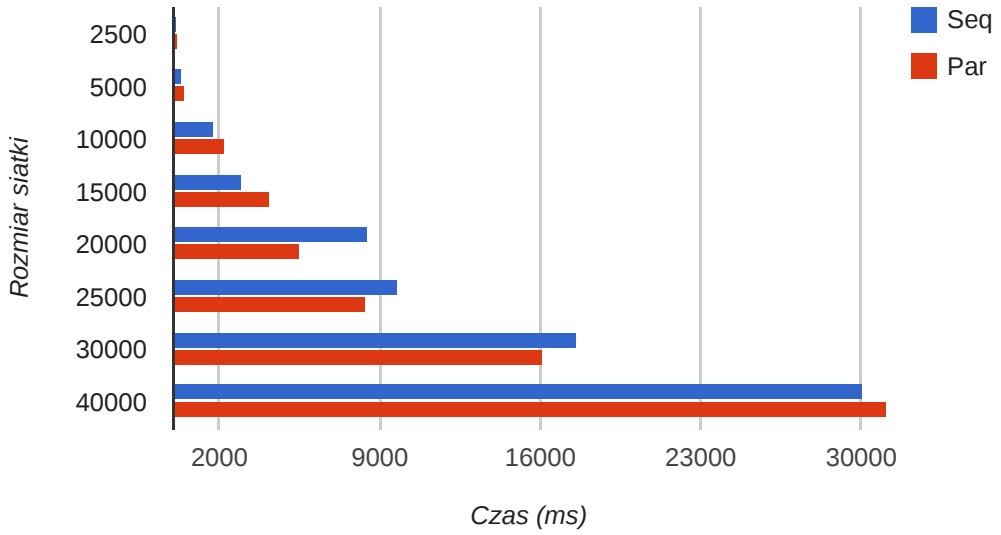


Rysunek 3.2: Czas działania oraz ilość iteracji wersji sekwencyjnej.

Widać dość wyraźne zaburzenie dla $N = 25000$, który wydaje się działać szybciej niż $N = 20000$. Warto jednak zauważyć, że ilość iteracji spada wraz ze wzrostem N, a zatem łączny czas sam w sobie nie pozwala na miarodajne porównywanie wyników. To co nas interesuje to średni czas na iterację. Przyjrzyjmy się mu, najlepiej od razu porównwując z wersją współbieżną w konfiguracji 1-1.

Wygląda to bardzo podobnie, ale nieco lepiej - teraz $N = 25000$ jest nieznacznie wolniejsze od $N = 20000$. Niemniej, wydaje się, że test dla $N = 25000$ działał w lepszych warunkach. Może węzeł na którym był liczony był mniej obciążony? Ponadto, rozwiązanie współbieżne czasem jest szybsze, a czasem wolniejsze, chociaż od strony kodu w zasadzie jest tym samym (posiada mały narzut potrzebny na sprawdzenie, że nie musi się z nikim komunikować).

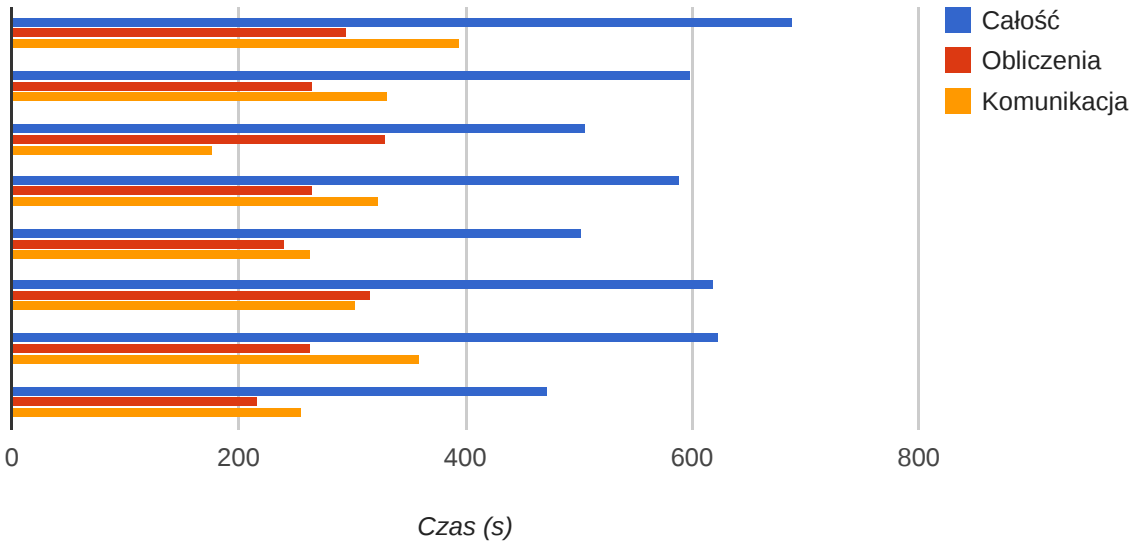
To prowadzi do pytania o powtarzalność testów.



Rysunek 3.3: Średni czas zużyty na iterację przez wersję sekwencyjną i współbieżną w konfiguracji 1-1.

3.3 Test „wariancji”

W celu sprawdzenia powtarzalności wyników uruchomiono test „`par-10000-4-2`” ośmiokrotnie.



Jak widać na wykresie, czas wykonania bardzo się waha. Różnica między najszybszym a najwolniejszym testem wynosi 216s! Co więcej, zarówno czas obliczeń jak i zużyty na przesyłanie wiadomości wahają się, co każe wnioskować, że nie jest to wina infrastruktury komunikacyjnej,

lecz faktyczna różnica w dostępnych zasobach CPU.

Prowadzi to do konkluzji, że jednokrotne wykonanie testów może dawać wyniki, które nie są w pełni porównywalne. Jednakże, w obliczu przeciążonego klastra nie pozostaje nam nic innego jak posiłkować się danymi, które zdołało się wygenerować.

3.4 Program współbieżny

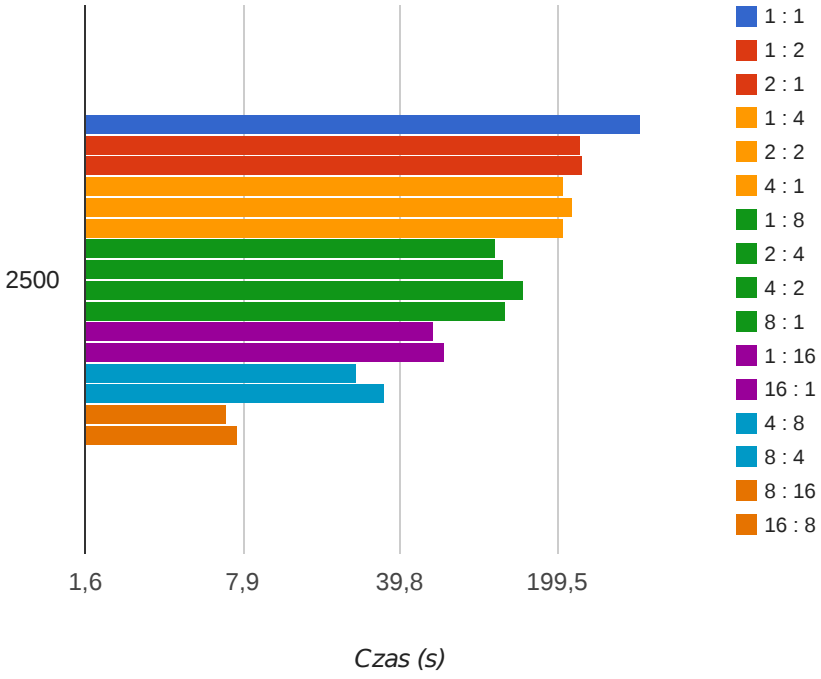
3.4.1 Całkowity czas pracy

Poniższe wykresy prezentują uśredniony czas iteracji dla różnych konfiguracji w rozwiązaniu współbieżnym. Skala jest logarytmiczna. Ilość iteracji została pominięta, gdyż jest dokładnie taka sama jak w rozwiązaniu sekwencyjnym.

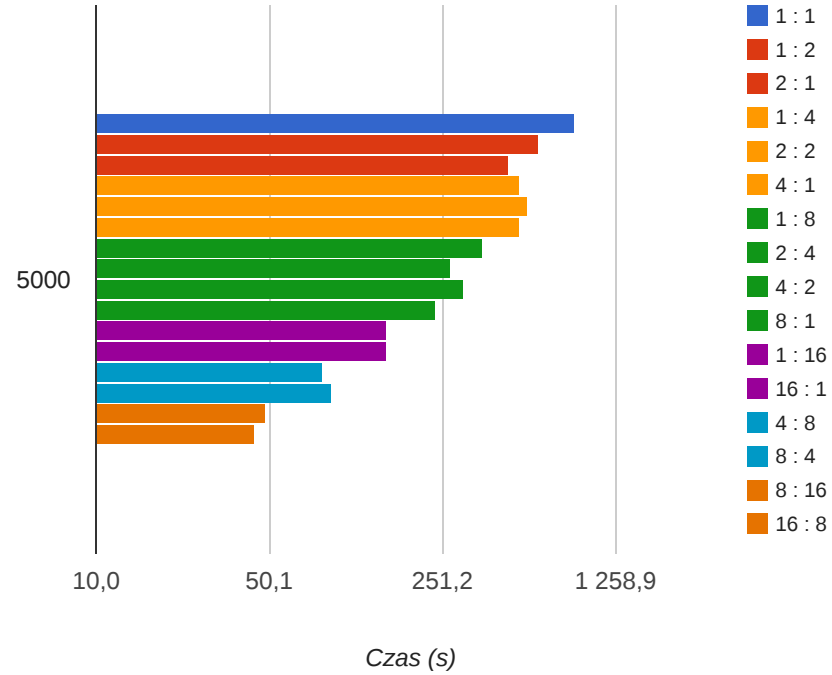
Zauważyć można dwie rzeczy.

Po pierwsze, czas spada mniej więcej proporcjonalnie do wzrostu ilości procesów, co świadczy o w miarę dobrym zrównolegleniu problemu.

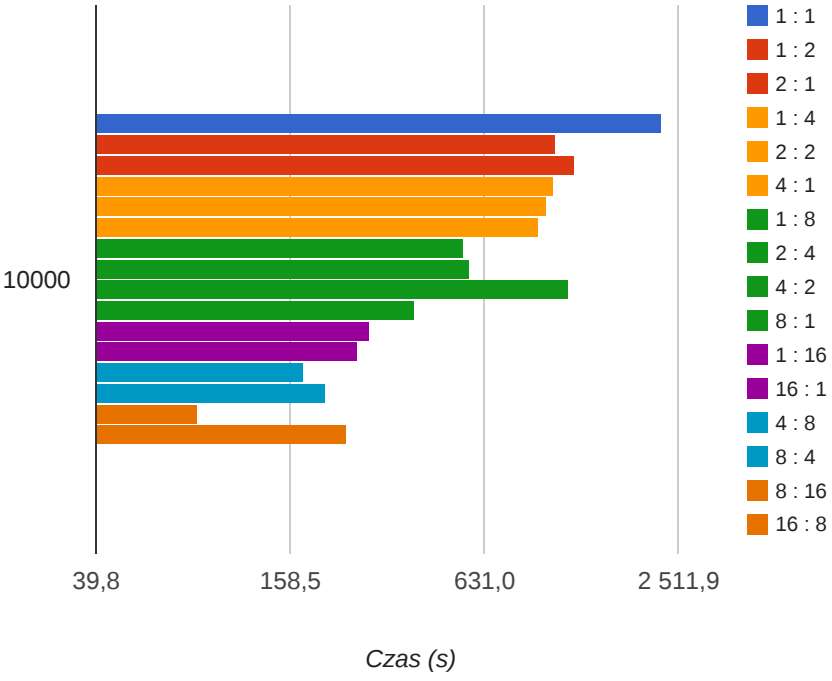
Po drugie - konfiguracje pracujące na mniejszej ilości maszyn, lecz większej ilości rdzeni mają przewagę nad konfiguracjami odwrotnymi. Wynika to najprawdopodobniej z większej lokalności w komunikacji, która rzadziej musi korzystać z łącz sieciowych. Przewaga ta rośnie wraz z ilością danych, które muszą wymienić ze sobą procesy.



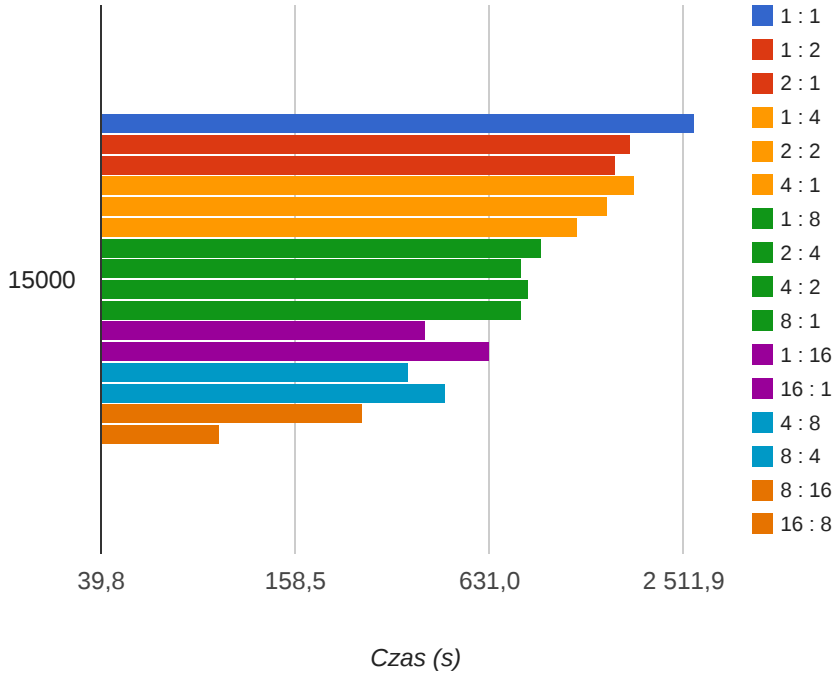
Różne konfiguracje współbieżne przy siatce o boku 2500.



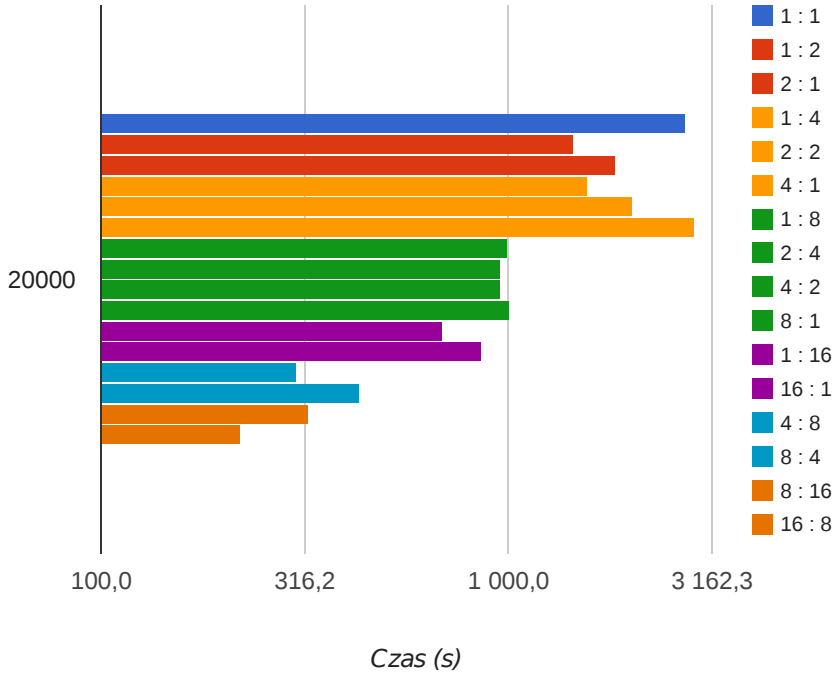
Różne konfiguracje współbieżne przy siatce o boku 5000.



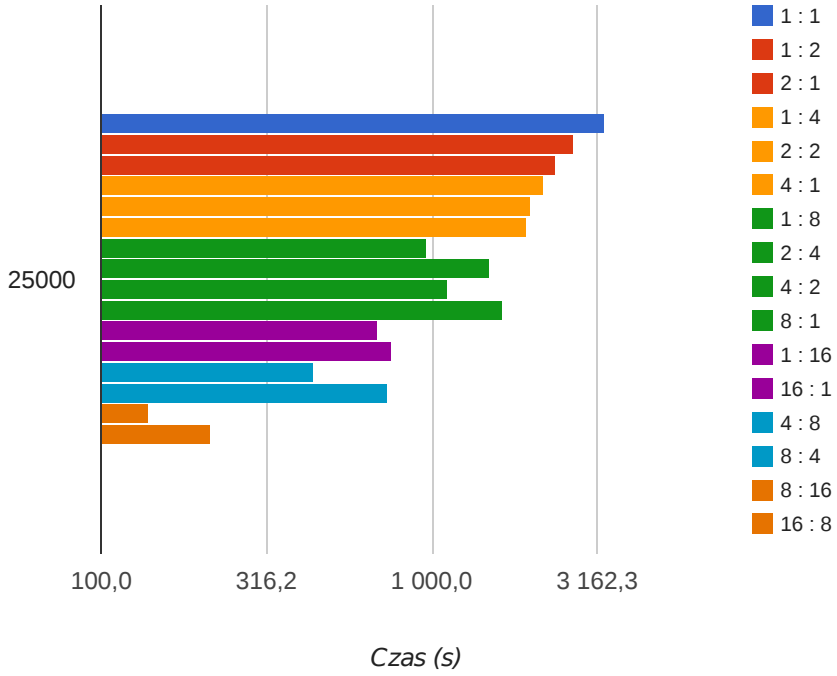
Różne konfiguracje współbieżne przy siatce o boku 10000.



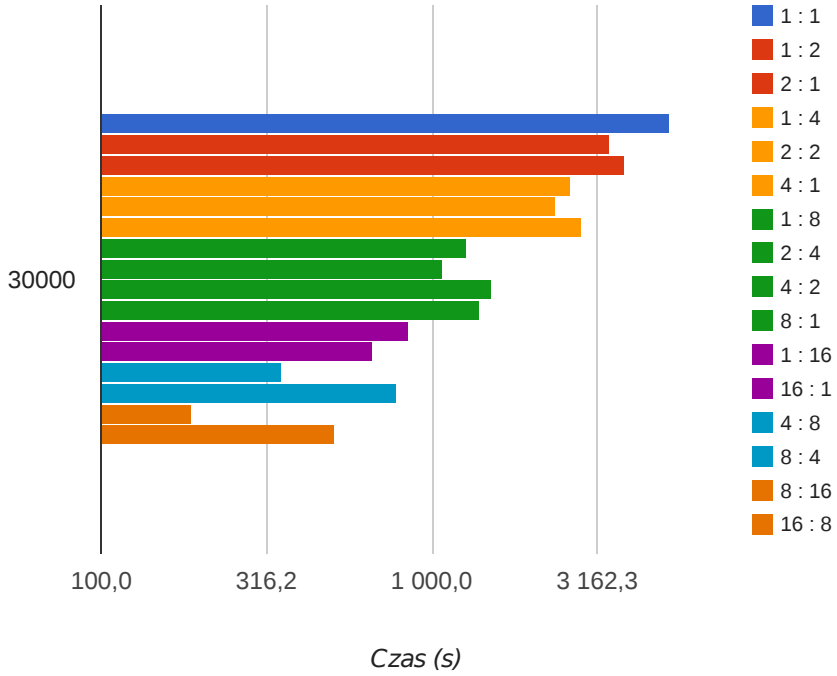
Różne konfiguracje współbieżne przy siatce o boku 15000.



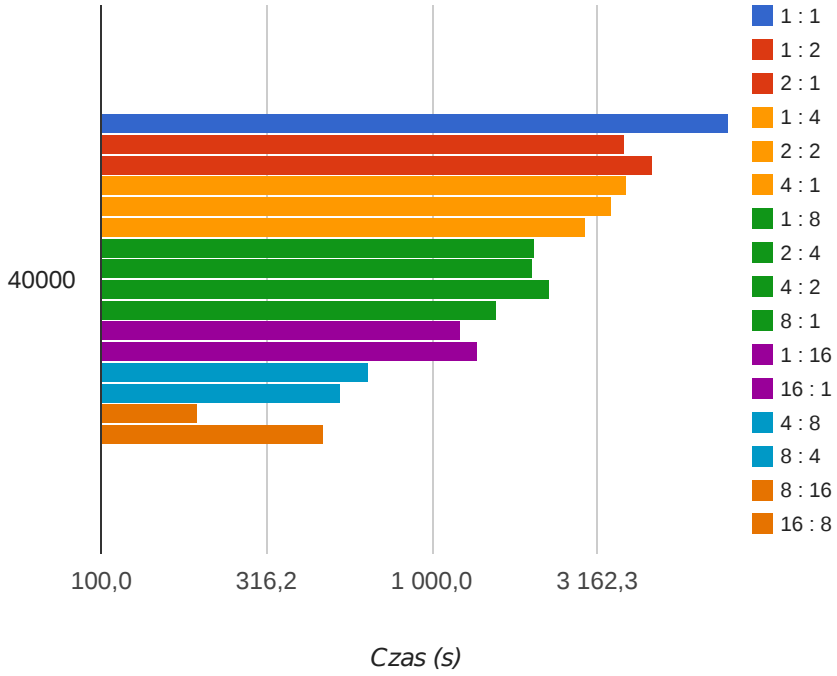
Różne konfiguracje współbieżne przy siatce o boku 20000.



Różne konfiguracje współbieżne przy siatce o boku 25000.



Różne konfiguracje współbieżne przy siatce o boku 30000.

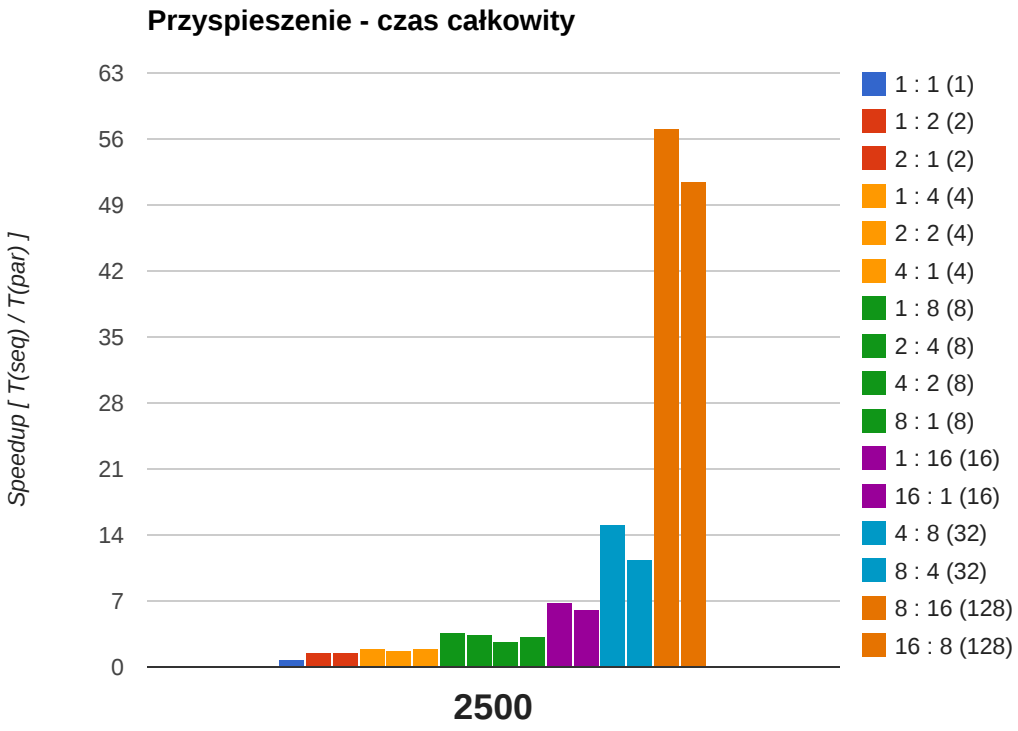


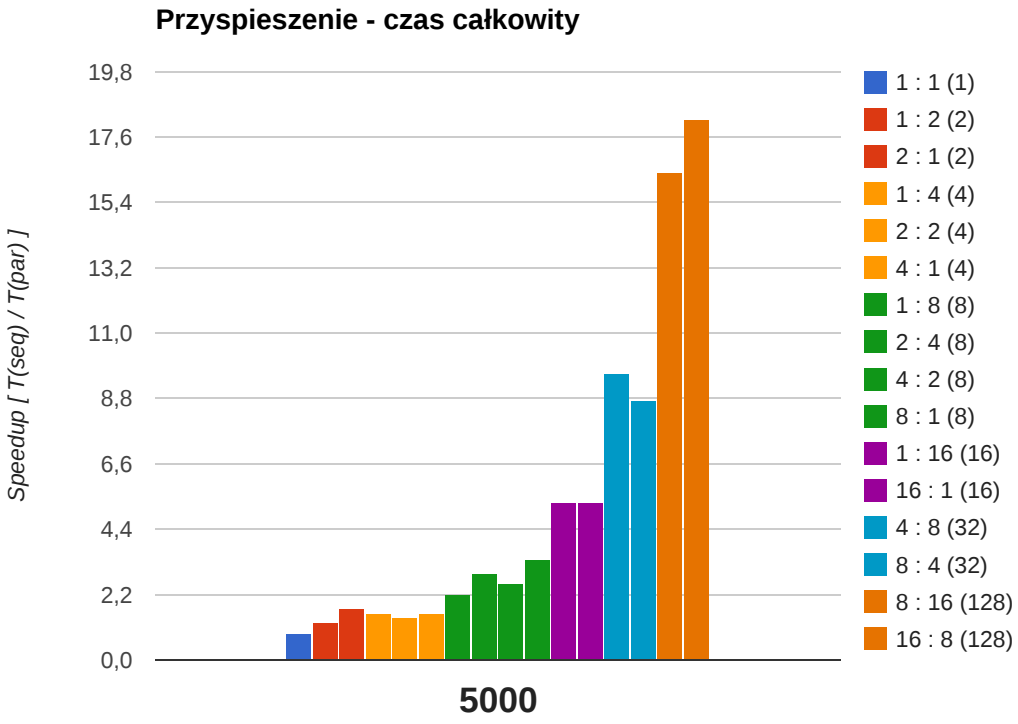
Różne konfiguracje współbieżne przy siatce o boku 40000.

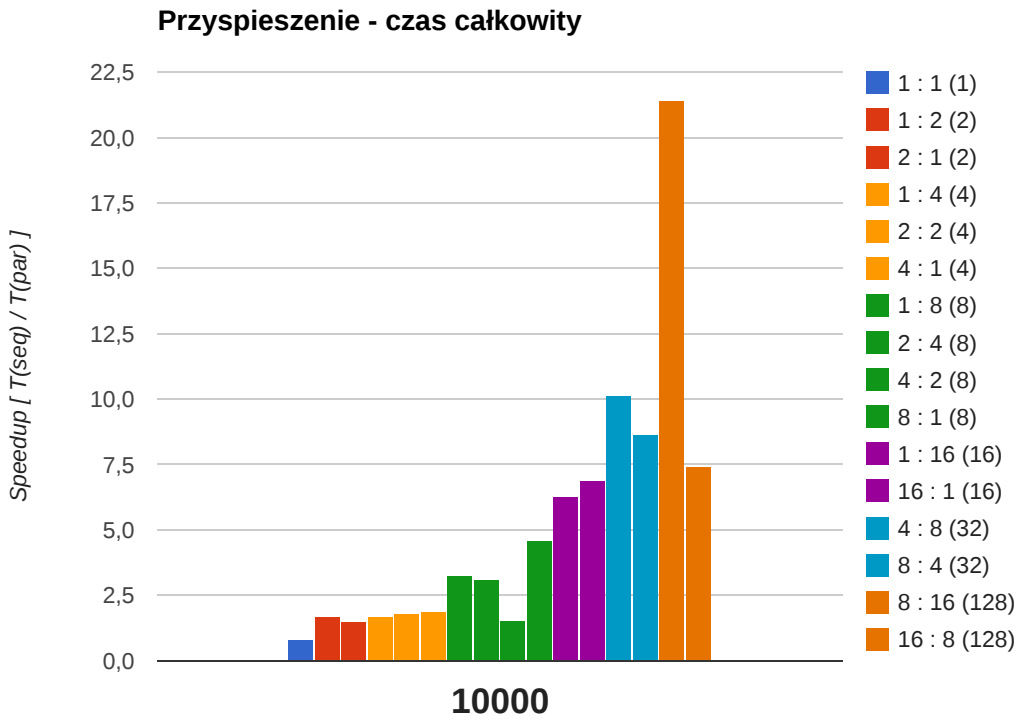
3.4.2 Speedup całkowitego czasu wykonania

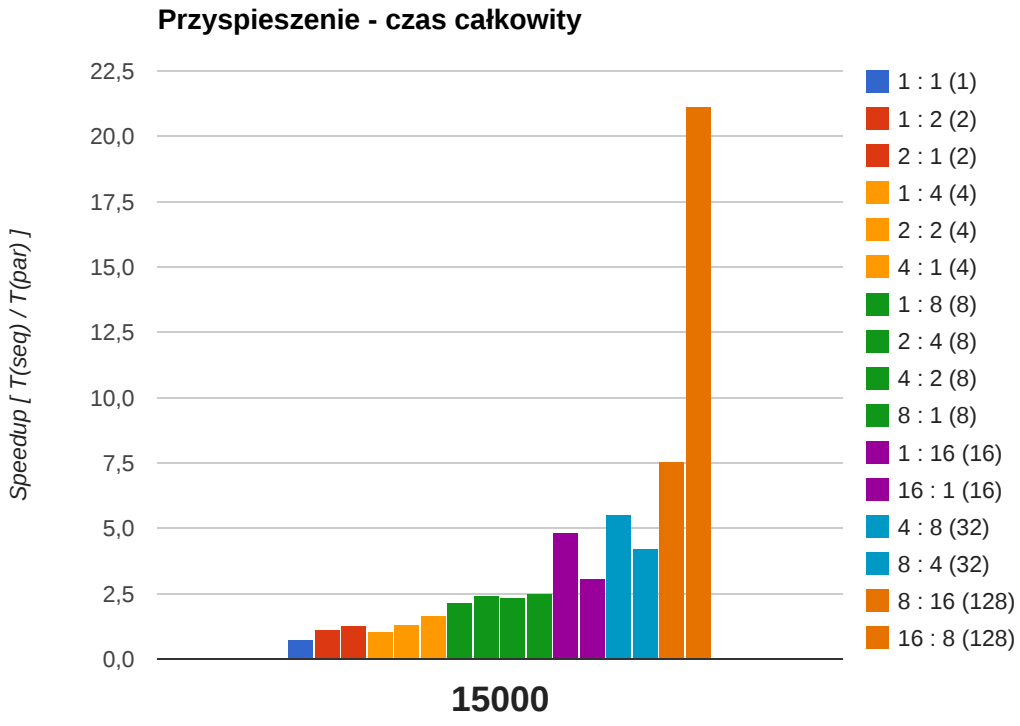
Przyspieszenie całkowite liczone jest jako stosunek czasu wykonania programu sekwencyjnego oraz współbieżnego. Liczby w nawiasach zamieszczone w legendzie to łączna ilość procesów biorąca udział w teście. Testy o tej samej liczności posiadają ten sam kolor słupków.

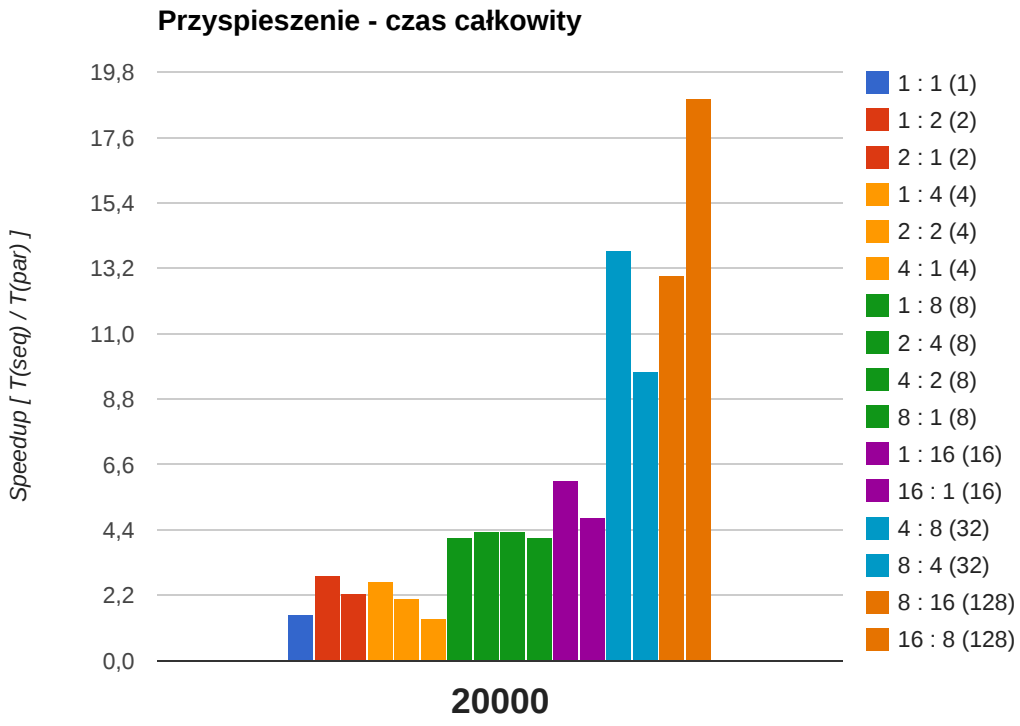
Wyraźnie widać, że przyspieszenie jest dalekie od ideału (będzie to lepiej widać potem, na wykresach efektywności). Ponownie widać spore wahania wyników oraz przewagę konfiguracji wielordzeniowych nad wielomaszynowymi.

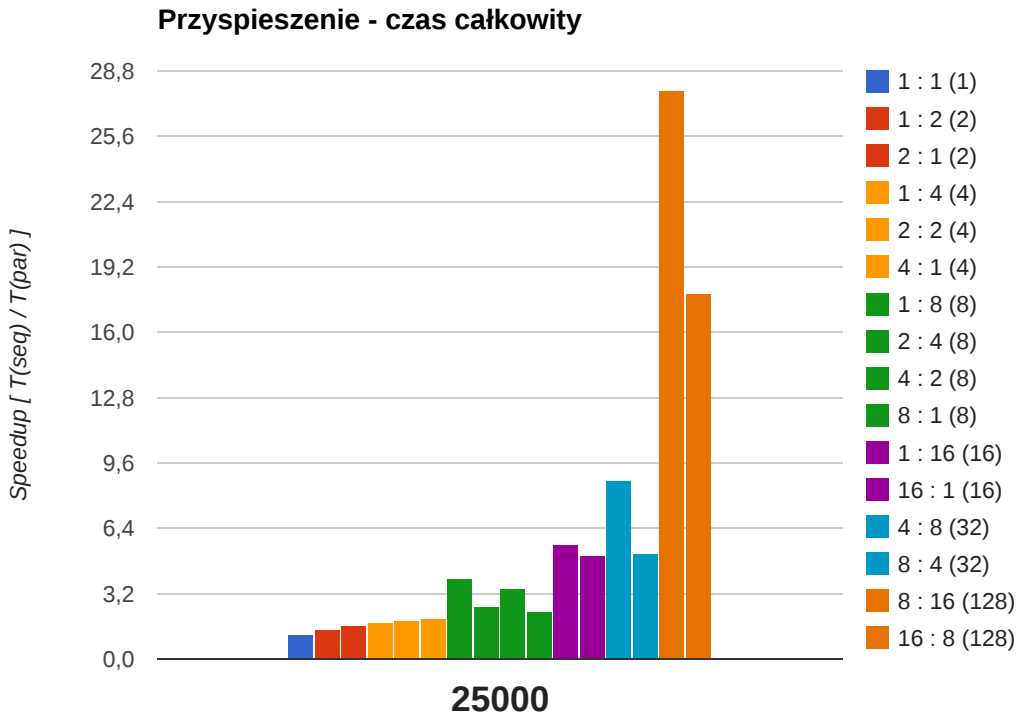

Przyspieszenie - czas całkowity
Speedup [T(seq) / T(par)]
63
56
49
42
35
28
21
14
7
0
2500
1 : 1 (1)
1 : 2 (2)
2 : 1 (2)
1 : 4 (4)
2 : 2 (4)
4 : 1 (4)
1 : 8 (8)
2 : 4 (8)
4 : 2 (8)
8 : 1 (8)
1 : 16 (16)
16 : 1 (16)
4 : 8 (32)
8 : 4 (32)
8 : 16 (128)
16 : 8 (128)

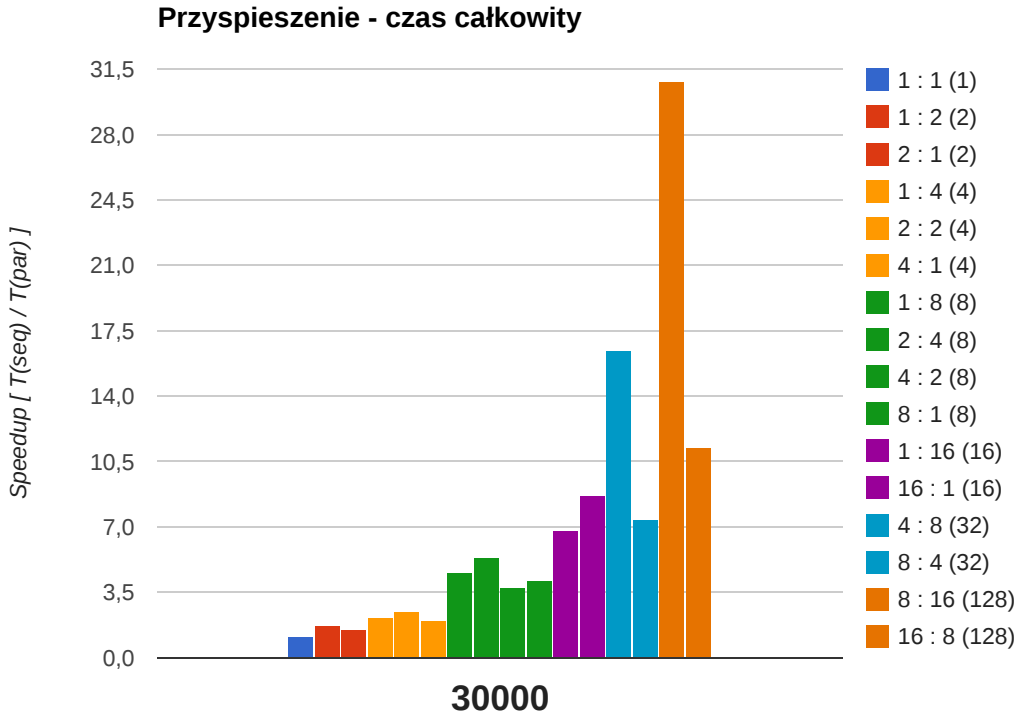

Przyspieszenie - czas całkowity
Speedup [T(seq) / T(par)]
19,8
17,6
15,4
13,2
11,0
8,8
6,6
4,4
2,2
0,0
5000
1 : 1 (1)
1 : 2 (2)
2 : 1 (2)
1 : 4 (4)
2 : 2 (4)
4 : 1 (4)
1 : 8 (8)
2 : 4 (8)
4 : 2 (8)
8 : 1 (8)
1 : 16 (16)
16 : 1 (16)
4 : 8 (32)
8 : 4 (32)
8 : 16 (128)
16 : 8 (128)

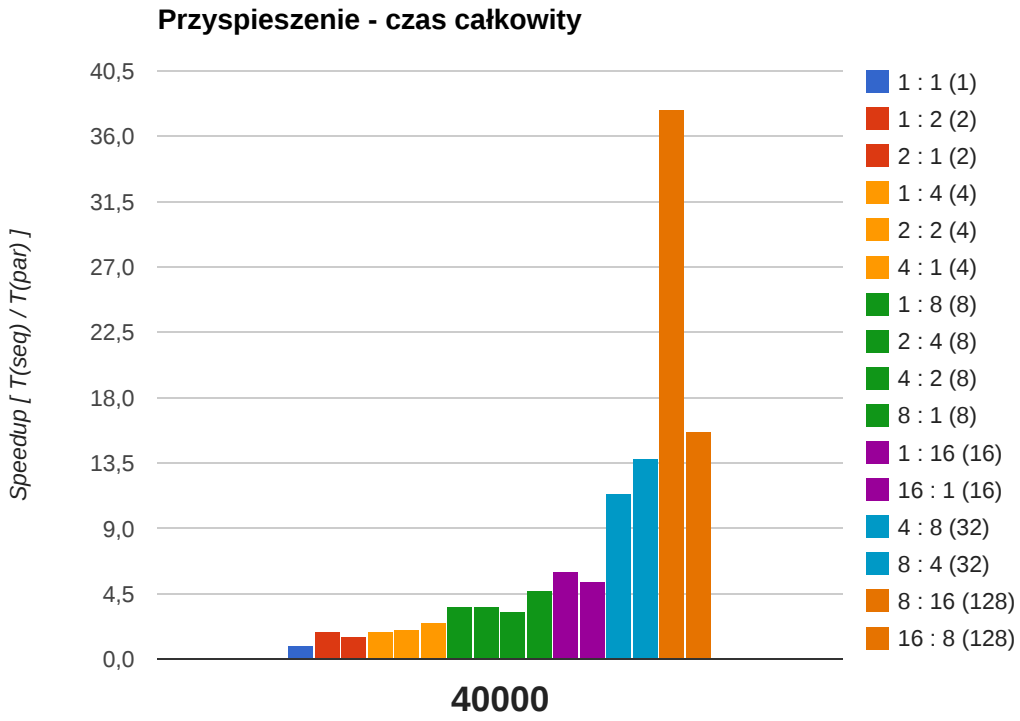

Przyspieszenie - czas całkowity
Speedup [T(seq) / T(par)]
22,5
20,0
17,5
15,0
12,5
10,0
7,5
5,0
2,5
0,0
10000
1 : 1 (1)
1 : 2 (2)
2 : 1 (2)
1 : 4 (4)
2 : 2 (4)
4 : 1 (4)
1 : 8 (8)
2 : 4 (8)
4 : 2 (8)
8 : 1 (8)
1 : 16 (16)
16 : 1 (16)
4 : 8 (32)
8 : 4 (32)
8 : 16 (128)
16 : 8 (128)


Przyspieszenie - czas całkowity
Speedup [T(seq) / T(par)]
22,5
20,0
17,5
15,0
12,5
10,0
7,5
5,0
2,5
0,0
15000
1 : 1 (1)
1 : 2 (2)
2 : 1 (2)
1 : 4 (4)
2 : 2 (4)
4 : 1 (4)
1 : 8 (8)
2 : 4 (8)
4 : 2 (8)
8 : 1 (8)
1 : 16 (16)
16 : 1 (16)
4 : 8 (32)
8 : 4 (32)
8 : 16 (128)
16 : 8 (128)


Przyspieszenie - czas całkowity
Speedup [T(seq) / T(par)]
19,8
17,6
15,4
13,2
11,0
8,8
6,6
4,4
2,2
0,0
20000
1 : 1 (1)
1 : 2 (2)
2 : 1 (2)
1 : 4 (4)
2 : 2 (4)
4 : 1 (4)
1 : 8 (8)
2 : 4 (8)
4 : 2 (8)
8 : 1 (8)
1 : 16 (16)
16 : 1 (16)
4 : 8 (32)
8 : 4 (32)
8 : 16 (128)
16 : 8 (128)


Przyspieszenie - czas całkowity
Speedup [T(seq) / T(par)]
28,8
25,6
22,4
19,2
16,0
12,8
9,6
6,4
3,2
0,0
25000
1 : 1 (1)
1 : 2 (2)
2 : 1 (2)
1 : 4 (4)
2 : 2 (4)
4 : 1 (4)
1 : 8 (8)
2 : 4 (8)
4 : 2 (8)
8 : 1 (8)
1 : 16 (16)
16 : 1 (16)
4 : 8 (32)
8 : 4 (32)
8 : 16 (128)
16 : 8 (128)

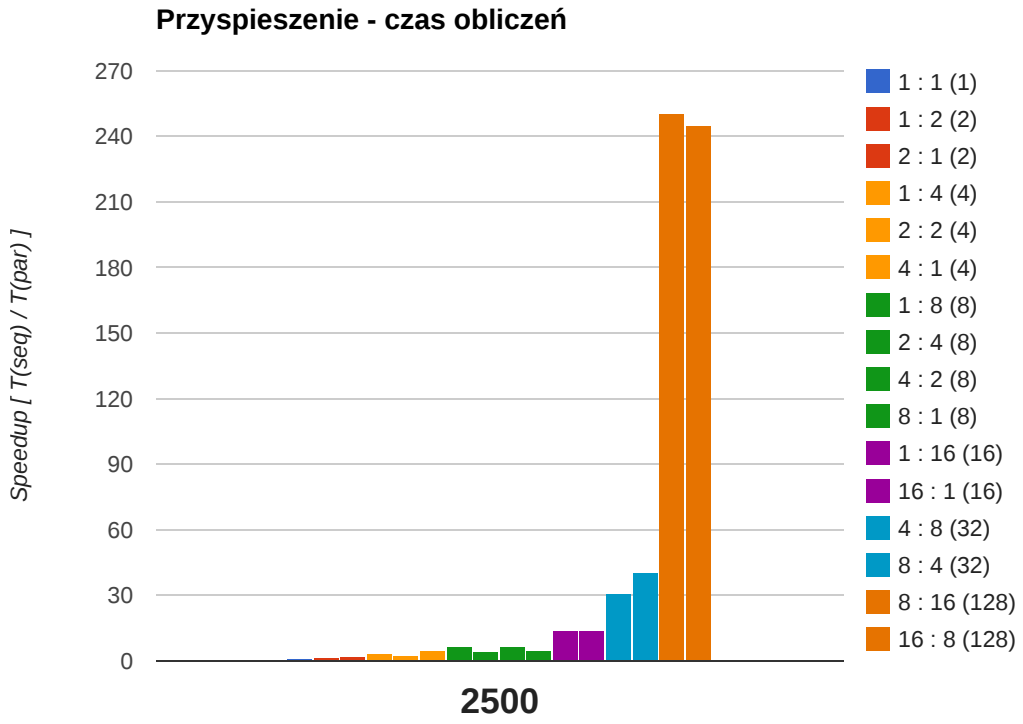


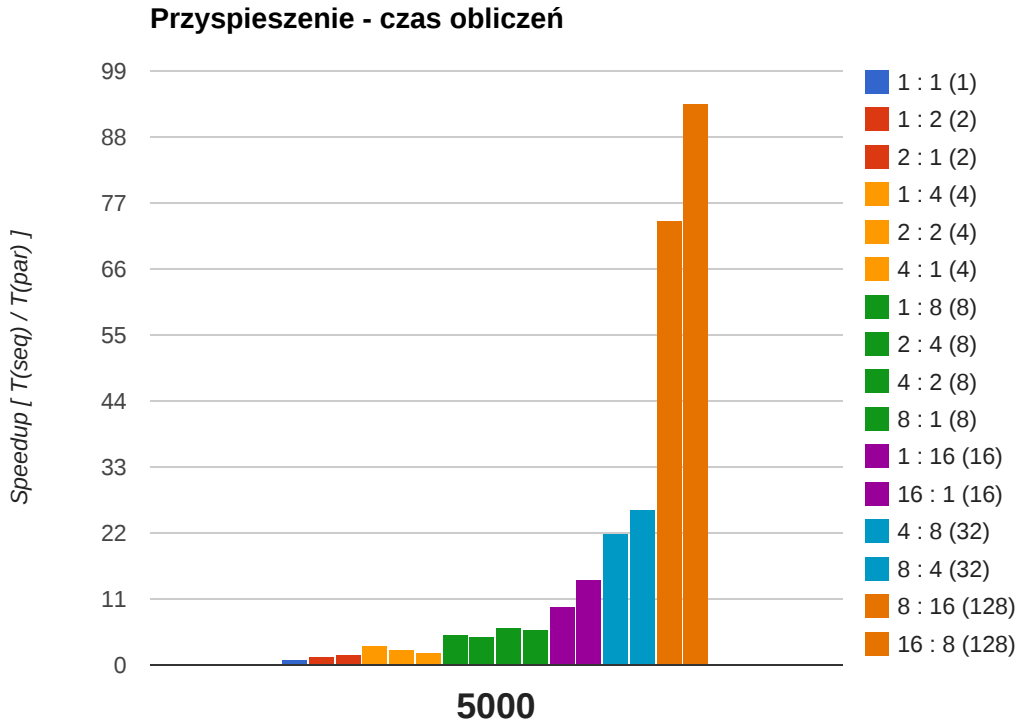


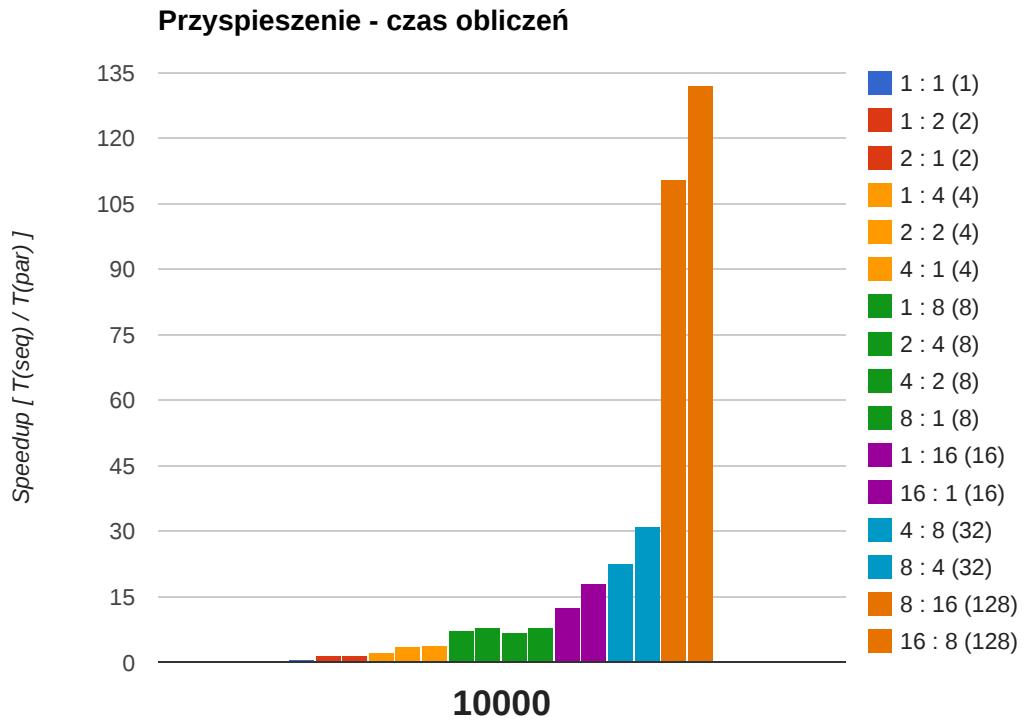
3.4.3 Speedup obliczeń

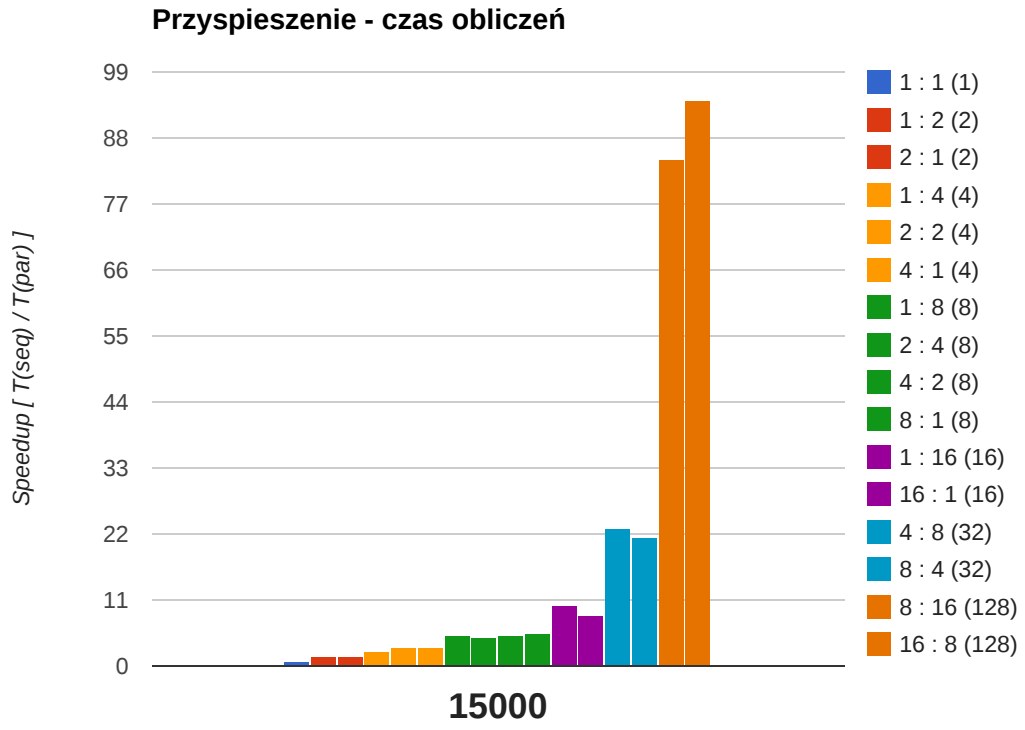
Nieco ciekawsze są wykresy uwzględniające jedynie czas zużyty na obliczenia. Wyniki są dużo bardziej stabilne i efektywniejsze. Ukazuje to jak dużym obciążeniem w programie rozproszo-

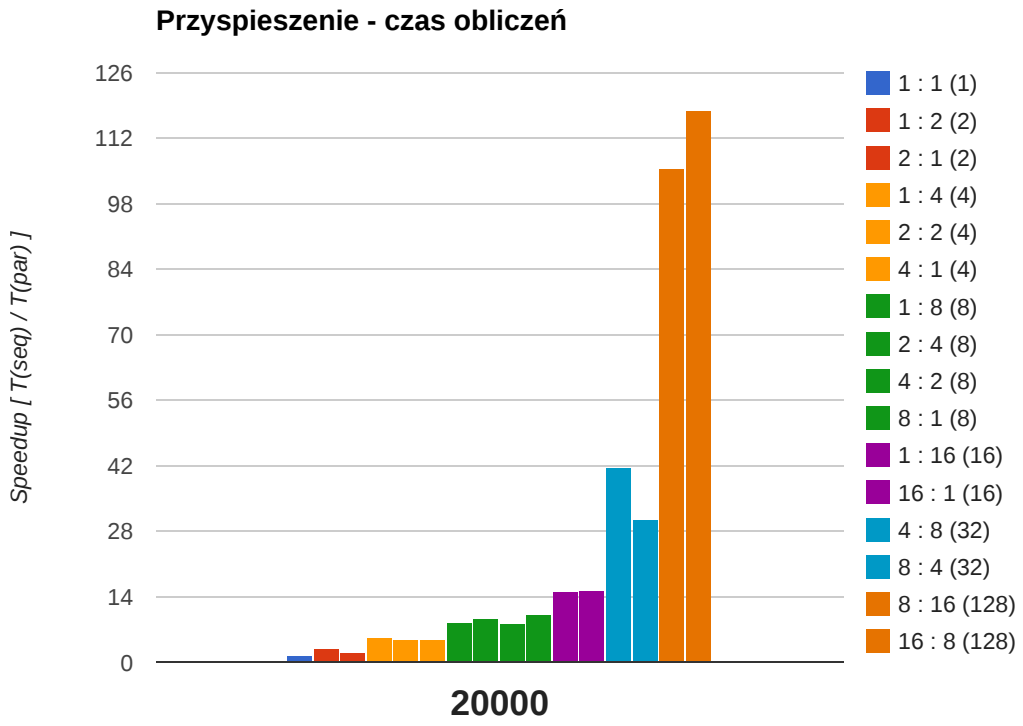
nym jest synchronizacja.

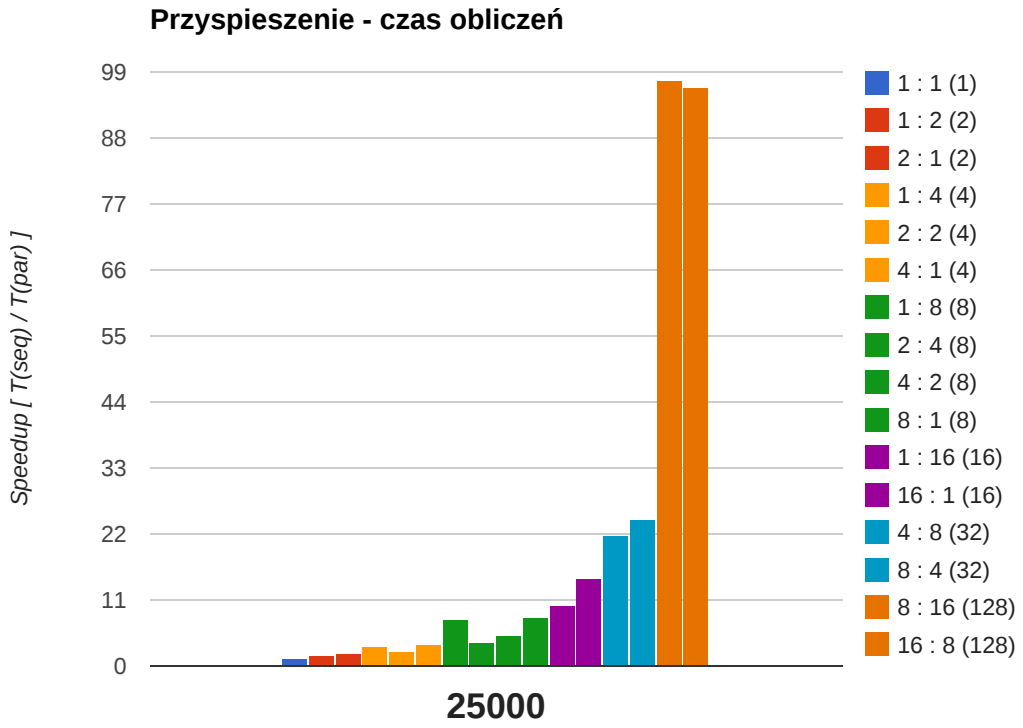


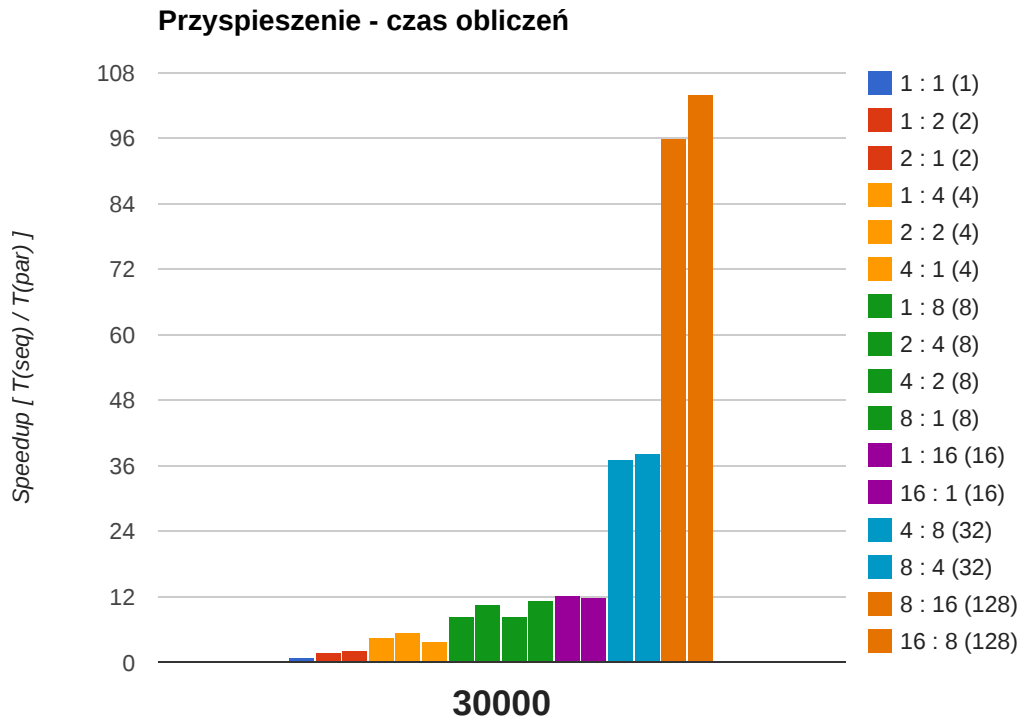


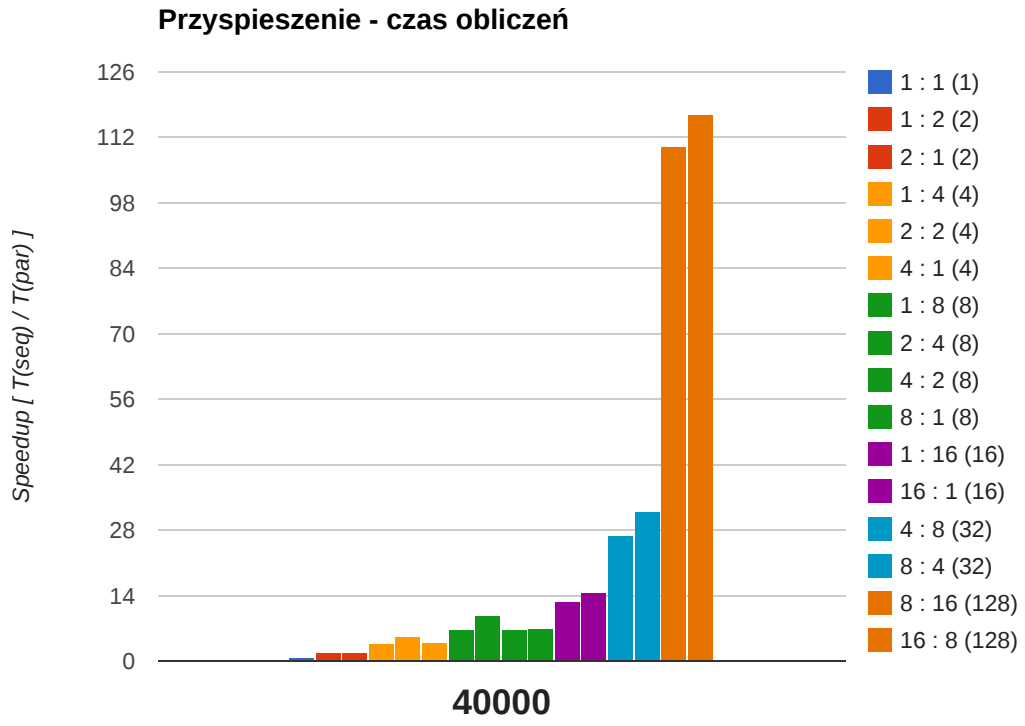

Przyspieszenie - czas obliczeń
Speedup [T(seq) / T(par)]
135
120
105
90
75
60
45
30
15
0
10000
1 : 1 (1)
1 : 2 (2)
2 : 1 (2)
1 : 4 (4)
2 : 2 (4)
4 : 1 (4)
1 : 8 (8)
2 : 4 (8)
4 : 2 (8)
8 : 1 (8)
1 : 16 (16)
16 : 1 (16)
4 : 8 (32)
8 : 4 (32)
8 : 16 (128)
16 : 8 (128)


Przyspieszenie - czas obliczeń
Speedup [T(seq) / T(par)]
99
88
77
66
55
44
33
22
11
0
15000
1 : 1 (1)
1 : 2 (2)
2 : 1 (2)
1 : 4 (4)
2 : 2 (4)
4 : 1 (4)
1 : 8 (8)
2 : 4 (8)
4 : 2 (8)
8 : 1 (8)
1 : 16 (16)
16 : 1 (16)
4 : 8 (32)
8 : 4 (32)
8 : 16 (128)
16 : 8 (128)


Przyspieszenie - czas obliczeń
Speedup [T(seq) / T(par)]
126
112
98
84
70
56
42
28
14
0
20000
1 : 1 (1)
1 : 2 (2)
2 : 1 (2)
1 : 4 (4)
2 : 2 (4)
4 : 1 (4)
1 : 8 (8)
2 : 4 (8)
4 : 2 (8)
8 : 1 (8)
1 : 16 (16)
16 : 1 (16)
4 : 8 (32)
8 : 4 (32)
8 : 16 (128)
16 : 8 (128)

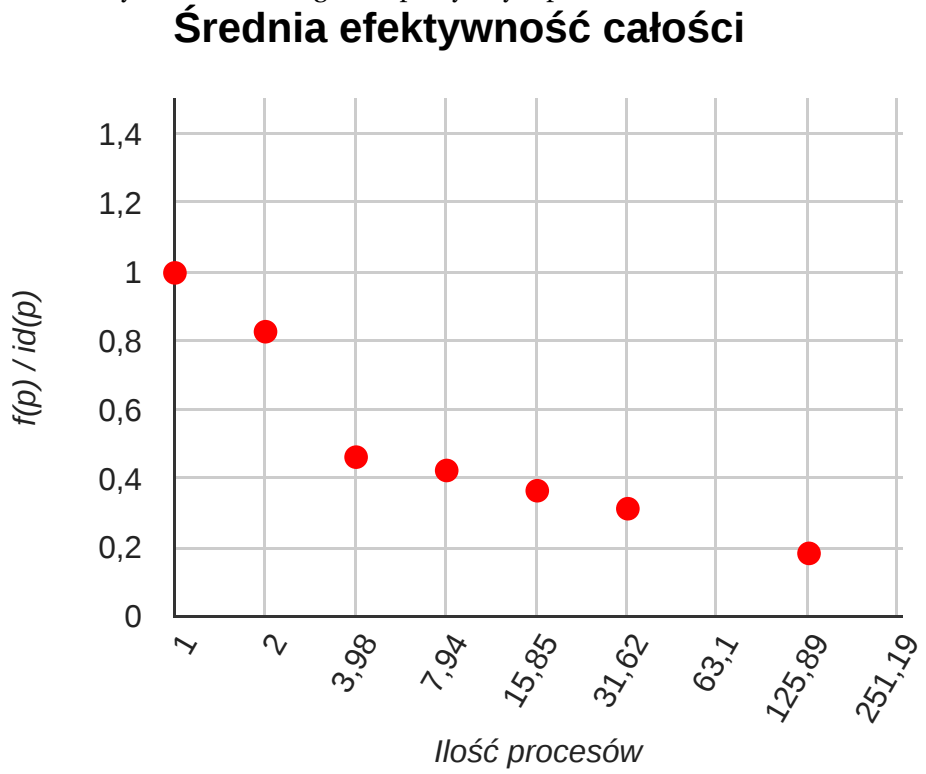

Przyspieszenie - czas obliczeń
Speedup [T(seq) / T(par)]
99
88
77
66
55
44
33
22
11
0
25000
1 : 1 (1)
1 : 2 (2)
2 : 1 (2)
1 : 4 (4)
2 : 2 (4)
4 : 1 (4)
1 : 8 (8)
2 : 4 (8)
4 : 2 (8)
8 : 1 (8)
1 : 16 (16)
16 : 1 (16)
4 : 8 (32)
8 : 4 (32)
8 : 16 (128)
16 : 8 (128)


Przyspieszenie - czas obliczeń
Speedup [T(seq) / T(par)]
108
96
84
72
60
48
36
24
12
0
30000
1 : 1 (1)
1 : 2 (2)
2 : 1 (2)
1 : 4 (4)
2 : 2 (4)
4 : 1 (4)
1 : 8 (8)
2 : 4 (8)
4 : 2 (8)
8 : 1 (8)
1 : 16 (16)
16 : 1 (16)
4 : 8 (32)
8 : 4 (32)
8 : 16 (128)
16 : 8 (128)


Przyspieszenie - czas obliczeń
Speedup [T(seq) / T(par)]
126
112
98
84
70
56
42
28
14
0
40000
1 : 1 (1)
1 : 2 (2)
2 : 1 (2)
1 : 4 (4)
2 : 2 (4)
4 : 1 (4)
1 : 8 (8)
2 : 4 (8)
4 : 2 (8)
8 : 1 (8)
1 : 16 (16)
16 : 1 (16)
4 : 8 (32)
8 : 4 (32)
8 : 16 (128)
16 : 8 (128)

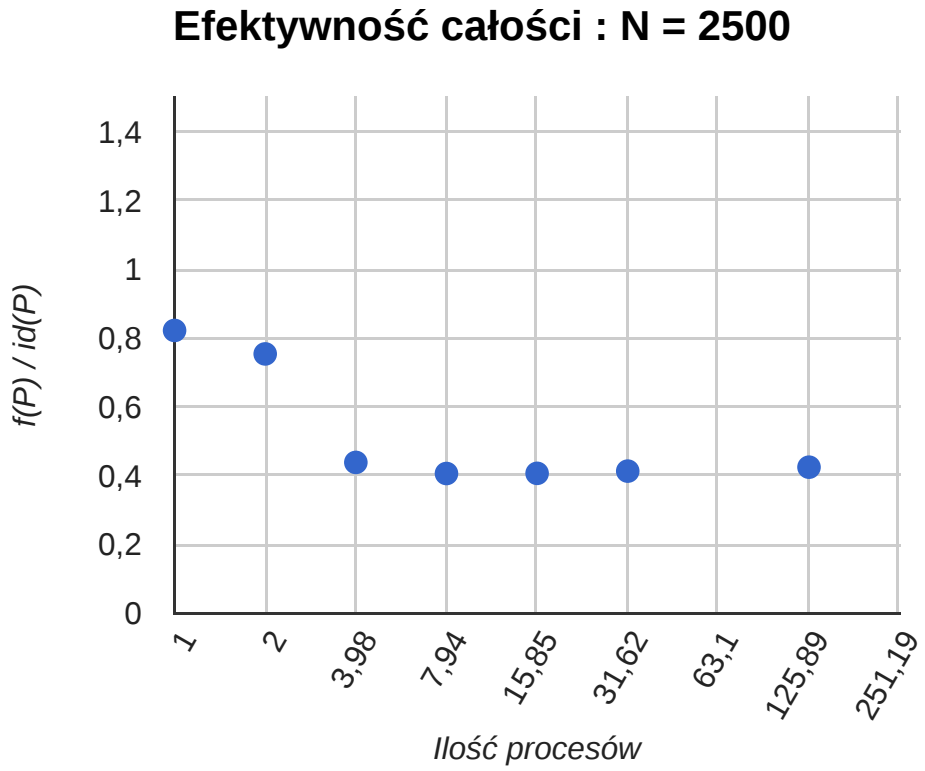
Poniższe wykresy ukazują średnią efektywność dla różnej ilości procesów biorących udział w teście. Jest ona wyliczona poprzez podzielenie średniego speedupu z wszystkich testów posiadających tą samą ilość procesów, przez „idealny” speedup, będący funkcją identycznościową od ilosci procesów.

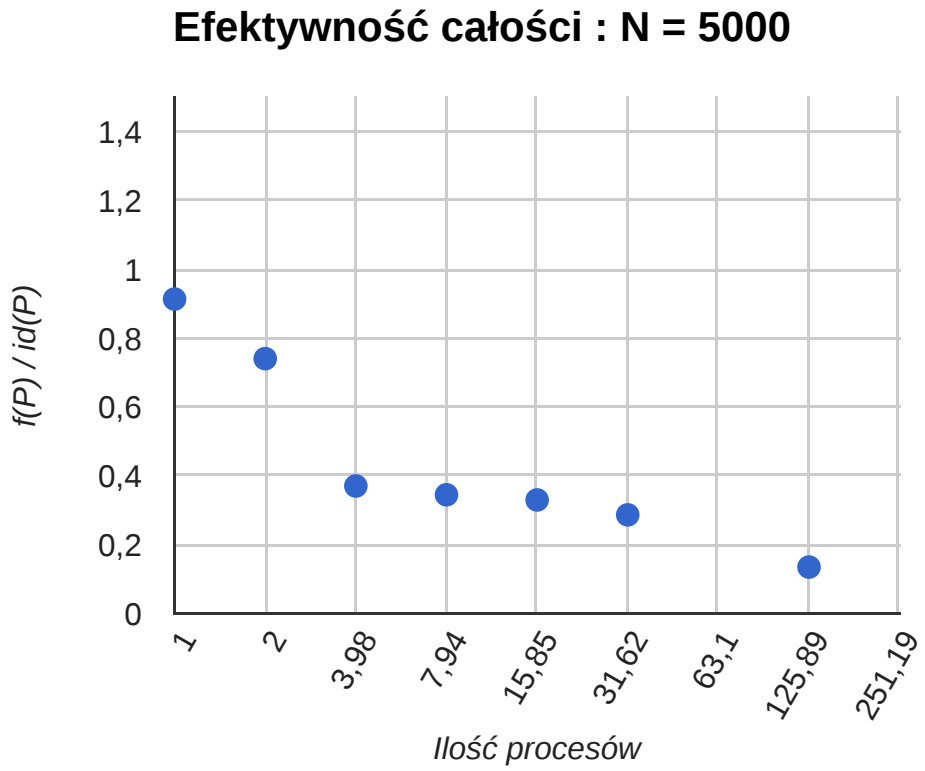
W przypadku łącznego czasu widać stopniową degradację jakości zrównoleglenia wraz ze wzrostem ilości procesów. Jest to spodowane rosnącym kosztem komunikacji. Ukazuje to kolejny wykres - efektywności obliczeń samych w sobie. Waha się on nieznacznie w przedziale $\{0.8, 1.0\}$, ale wygląda to jak zaburzenia spowodowane dużą wariancją czasów, a nie konstrukcją programu. W szczególności, efektywność wydaje się nie spadać wraz ze wzrostem ilości procesów, tylko właśnie „drgać” w powyższym przedziale.



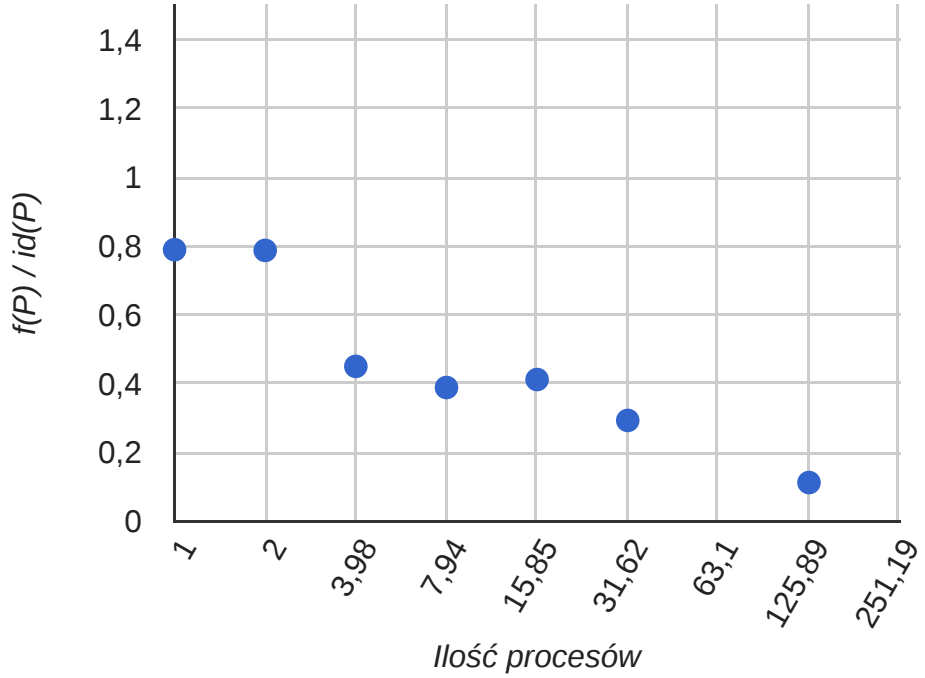
3.6 Efektywność, szczegółowo

Poniższe wykresy prezentują efektywność dla poszczególnych rozmiarów siatki. W zasadzie nie mówią wiele więcej niż wykres uśredniony, poza faktem, że przypadek $N = 2500$ jest niemiarodajny dla większych ilości procesów. Wynika to najprawdopodobniej z faktu, że czas potrzebny na czynności przygotowawcze do obliczeń jest wtedy bliski czasowi potrzebne na samo obliczenie i zaciera wynik.

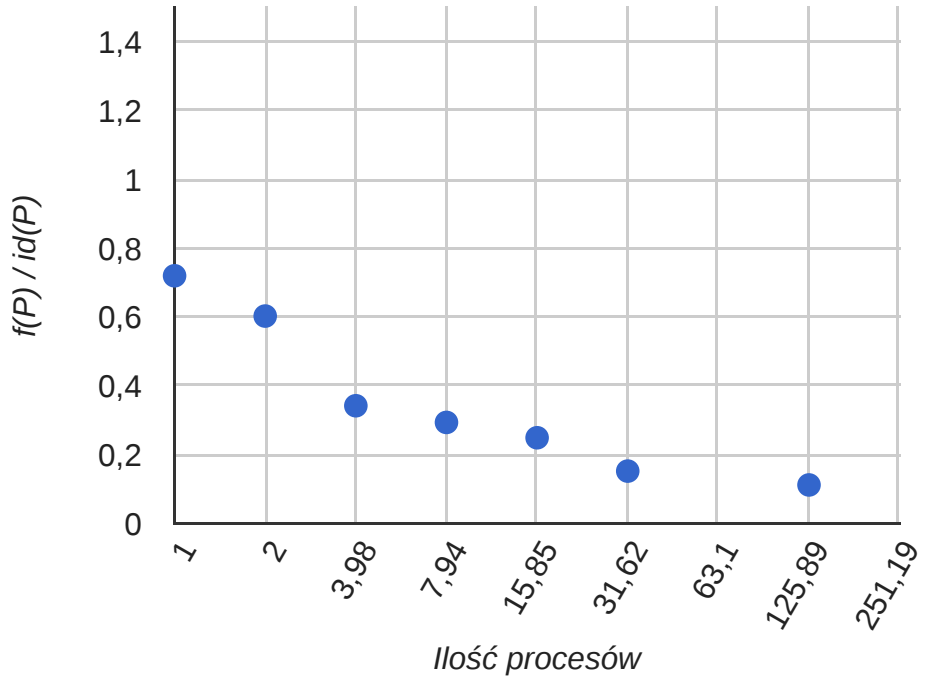

Efektywność całości : N = 2500
1,4
1,2
1
0,8
0,6
0,4
0,2
0
f(P) / id(P)
1
2
3,98
7,94
15,85
31,62
63,1
125,89
251,19
Ilość procesów

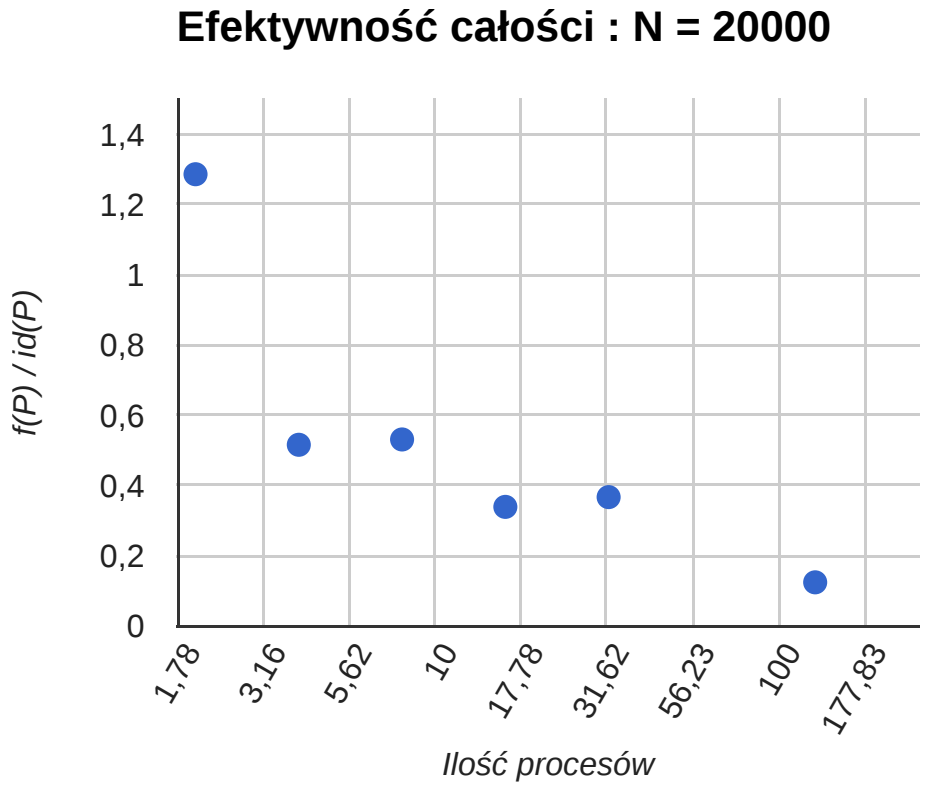

Efektywność całości : N = 5000
1,4
1,2
1
0,8
0,6
0,4
0,2
0
f(P) / id(P)
1
2
3,98
7,94
15,85
31,62
63,1
125,89
251,19
Ilość procesów

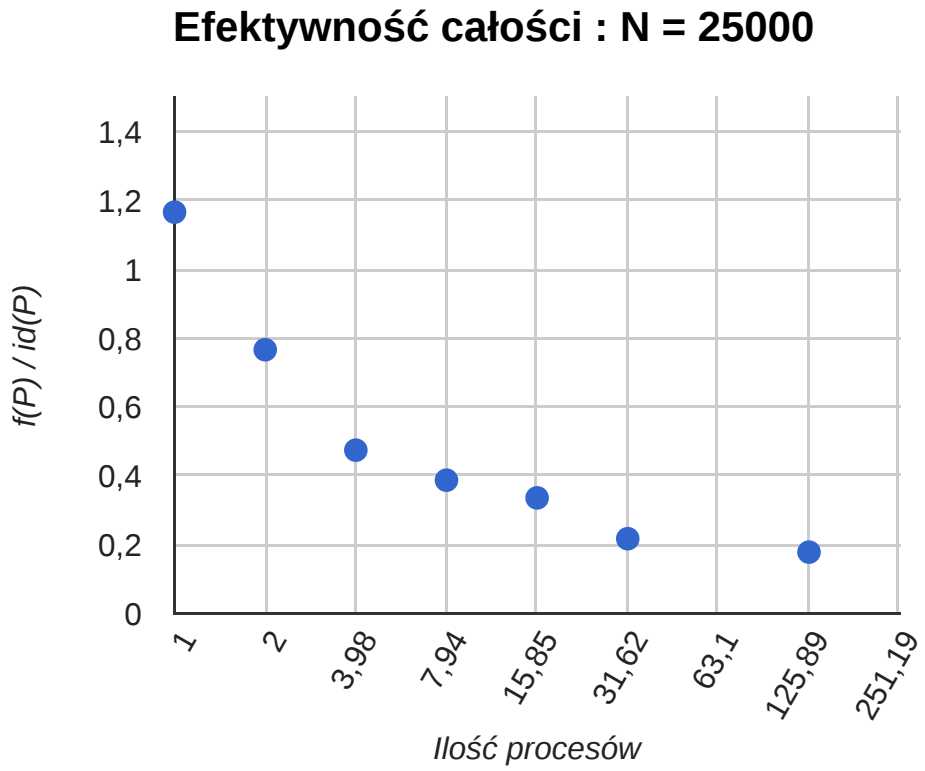
Efektywność całości : N = 10000

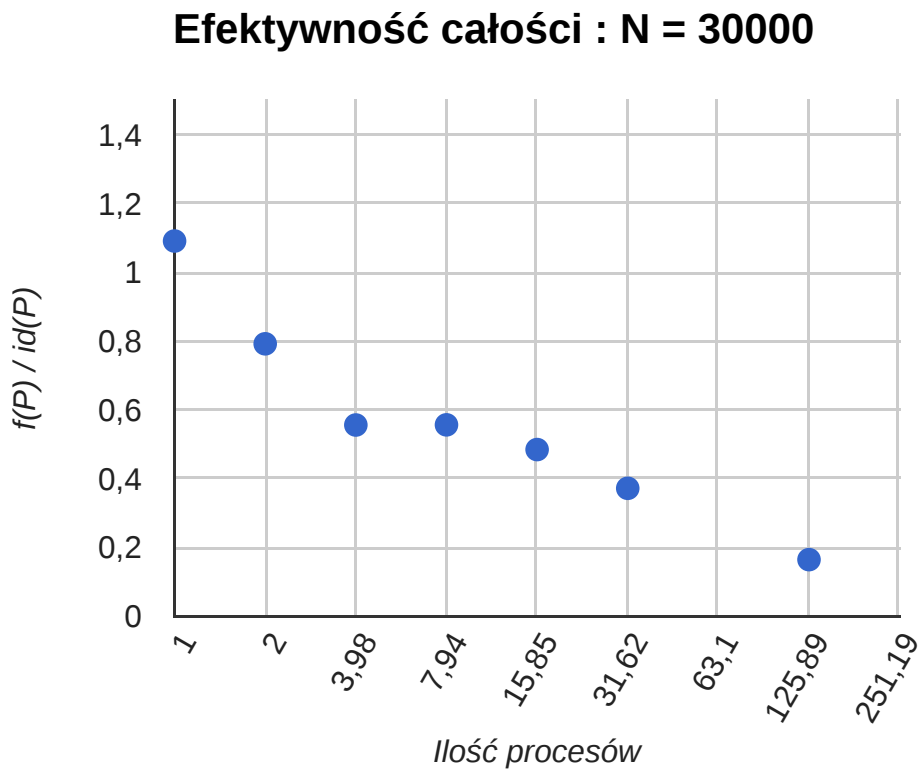


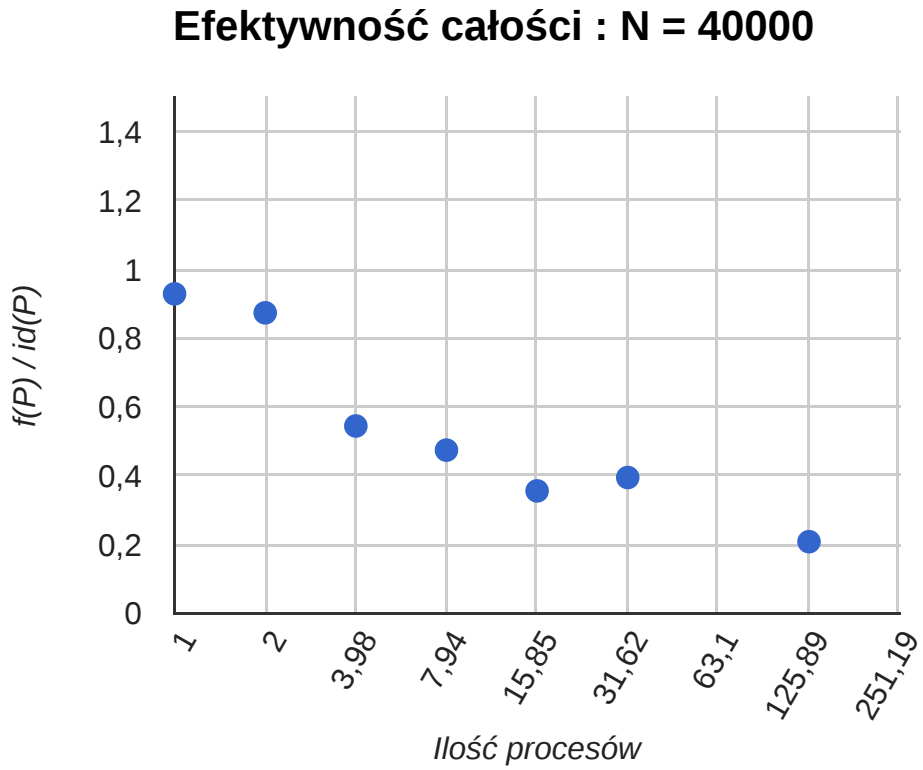
Efektywność całości : N = 15000

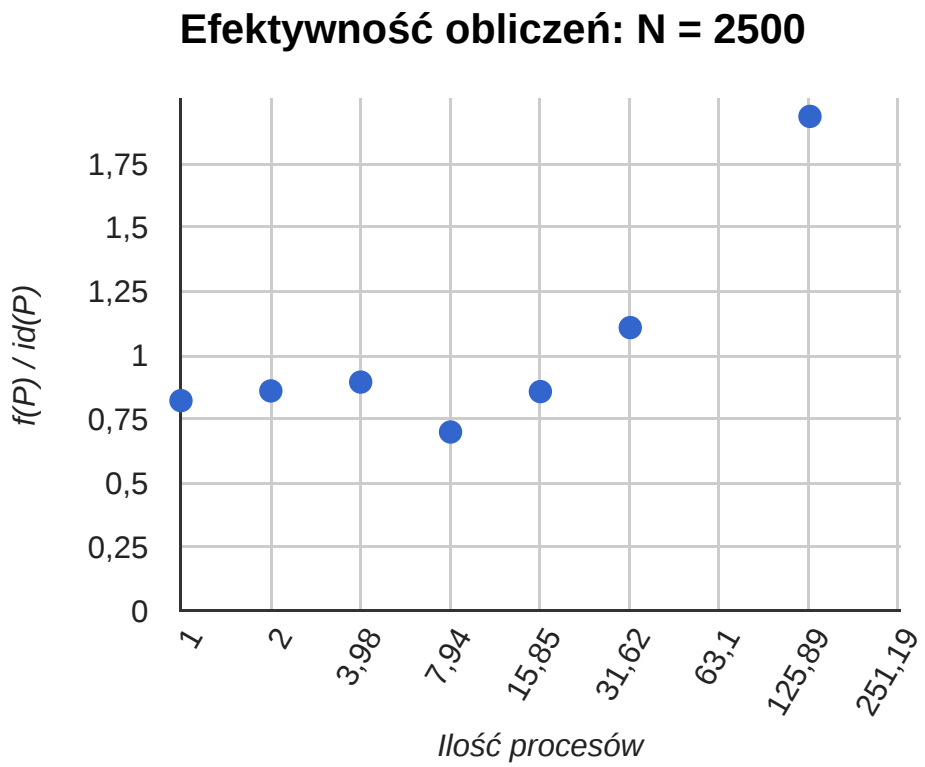


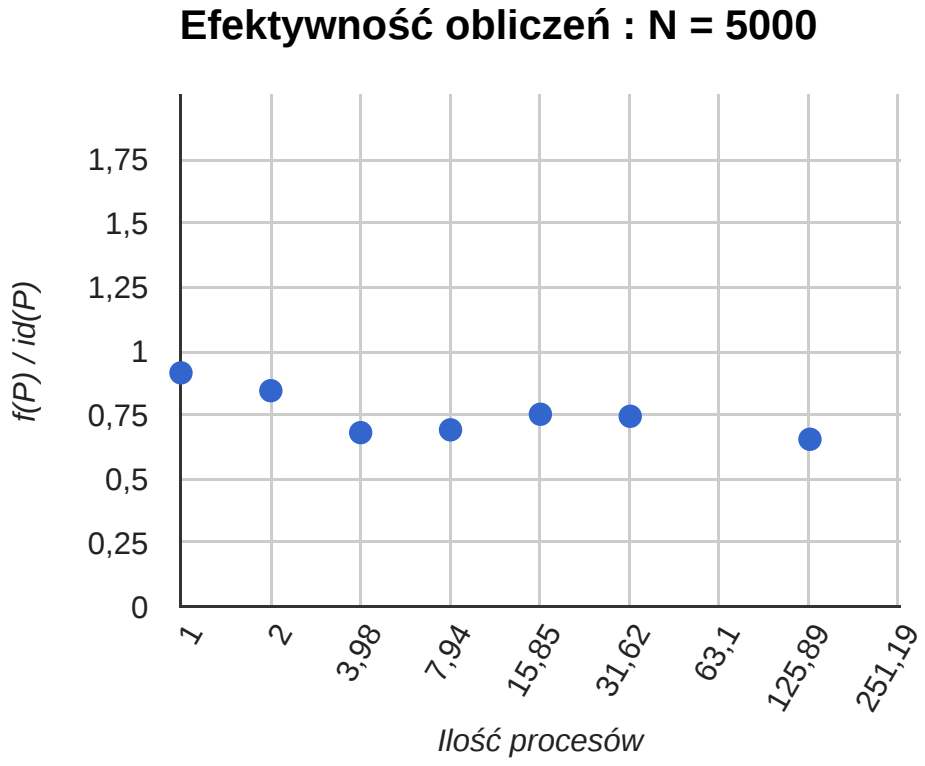

Efektywność całości : N = 20000
f(P) / id(P)
1,4
1,2
1
0,8
0,6
0,4
0,2
0
1,78
3,16
5,62
10
17,78
31,62
56,23
100
177,83
Ilość procesów

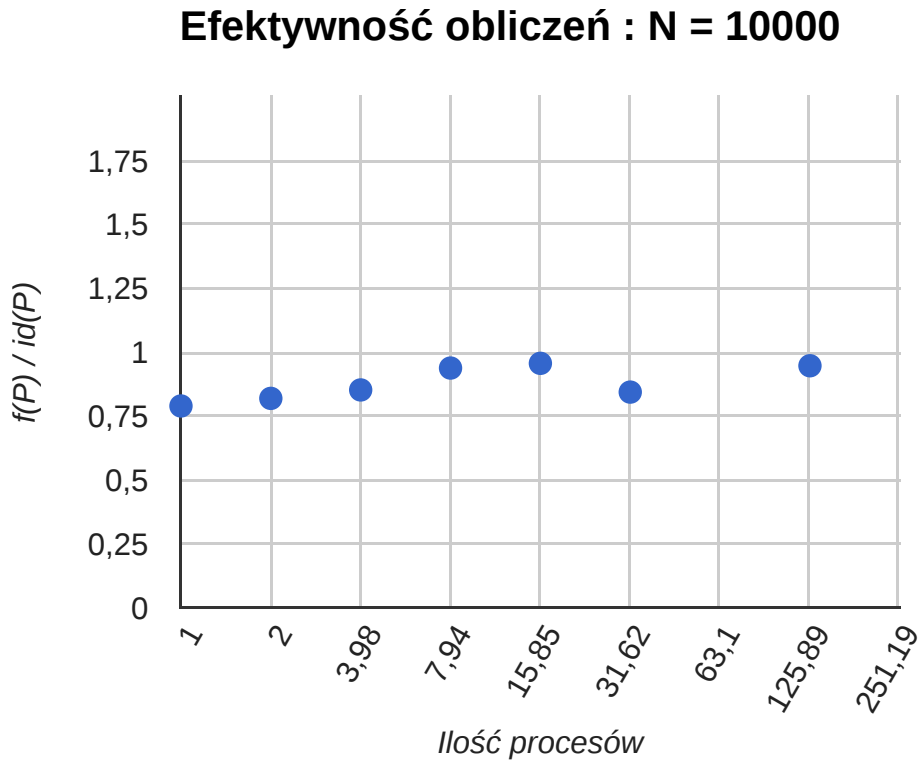

Efektywność całości : N = 25000
f(P) / id(P)
1,4
1,2
1
0,8
0,6
0,4
0,2
0
1
2
3,98
7,94
15,85
31,62
63,1
125,89
251,19
Ilość procesów

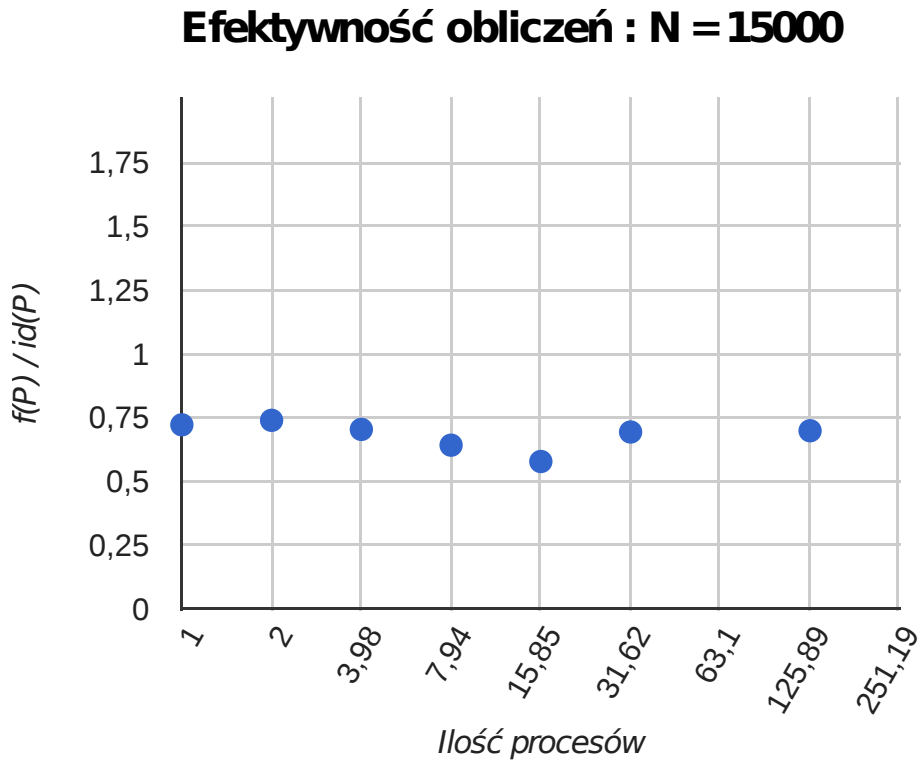

Efektywność całości : N = 30000
1,4
1,2
1
0,8
0,6
0,4
0,2
0
f(P) / id(P)
1
2
3,98
7,94
15,85
31,62
63,1
125,89
251,19
Ilość procesów


Efektywność całości : N = 40000
1,4
1,2
1
0,8
0,6
0,4
0,2
0
f(P) / id(P)
1
2
3,98
7,94
15,85
31,62
63,1
125,89
251,19
Ilość procesów

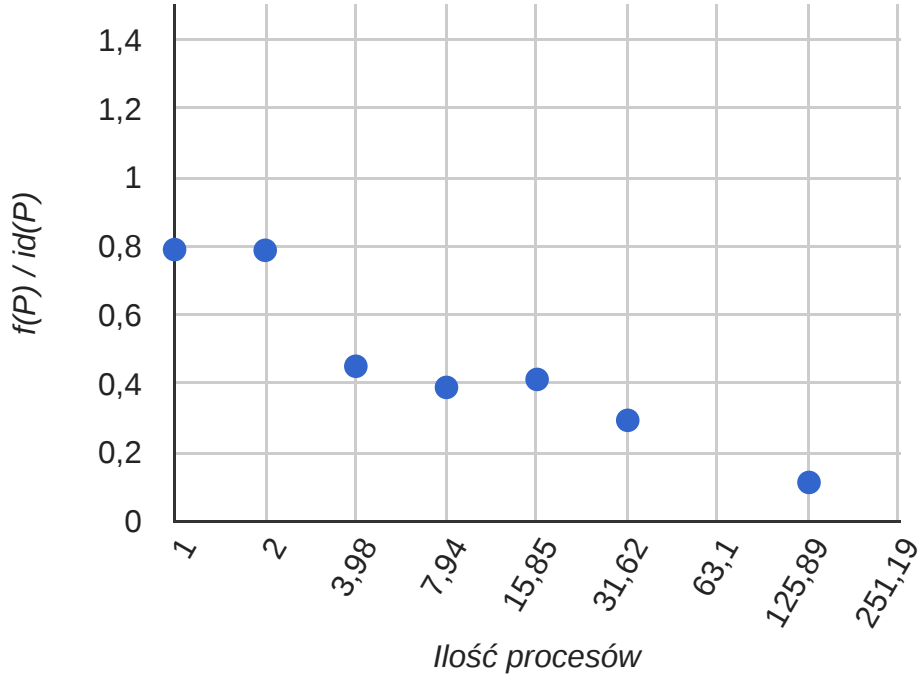

Efektywność obliczeń: N = 2500
f(P) / id(P)
1,75
1,5
1,25
1
0,75
0,5
0,25
0
1
2
3,98
7,94
15,85
31,62
63,1
125,89
251,19
Ilość procesów


Efektywność obliczeń : N = 5000
f(P) / id(P)
1,75
1,5
1,25
1
0,75
0,5
0,25
0
1
2
3,98
7,94
15,85
31,62
63,1
125,89
251,19
Ilość procesów

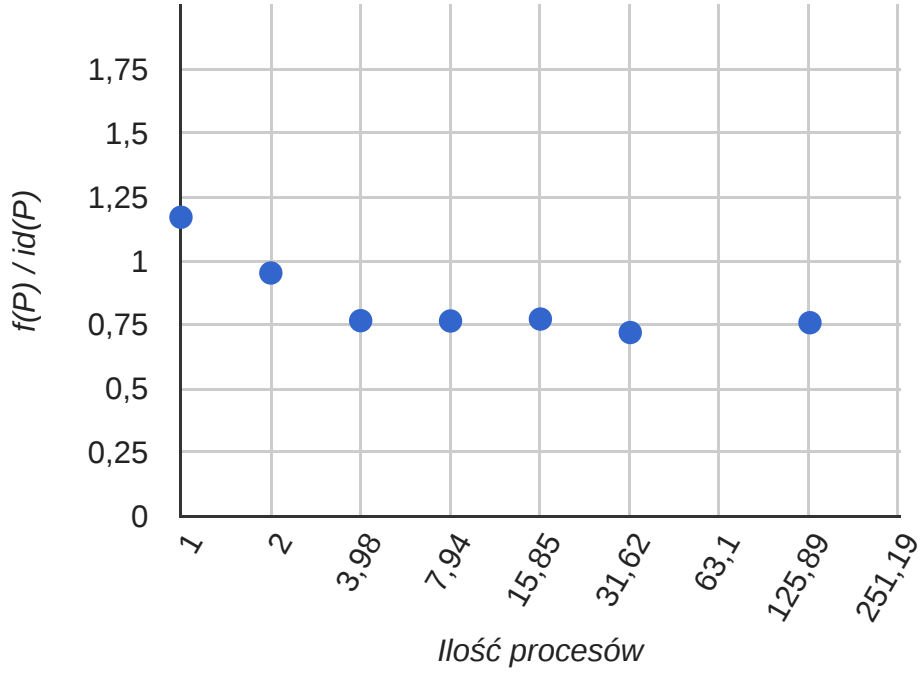

Efektywność obliczeń : N = 10000
f(P) / id(P)
1,75
1,5
1,25
1
0,75
0,5
0,25
0
1
2
3,98
7,94
15,85
31,62
63,1
125,89
251,19
Ilość procesów


Efektywność obliczeń : N = 15000
f(P) / id(P)
1,75
1,5
1,25
1
0,75
0,5
0,25
0
1
2
3,98
7,94
15,85
31,62
63,1
125,89
251,19
Ilość procesów

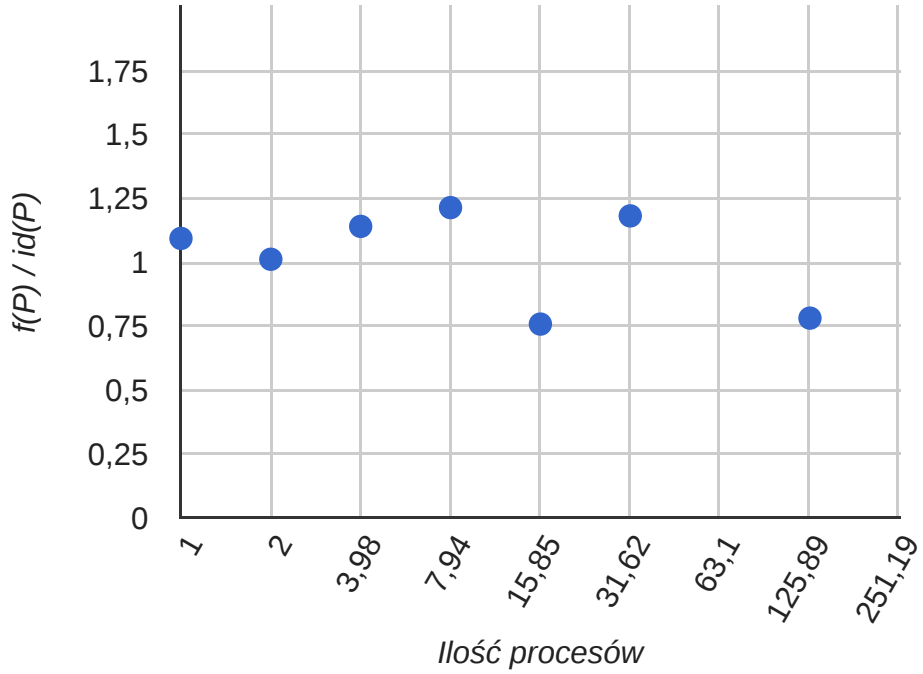
Efektywność całości : N = 20000



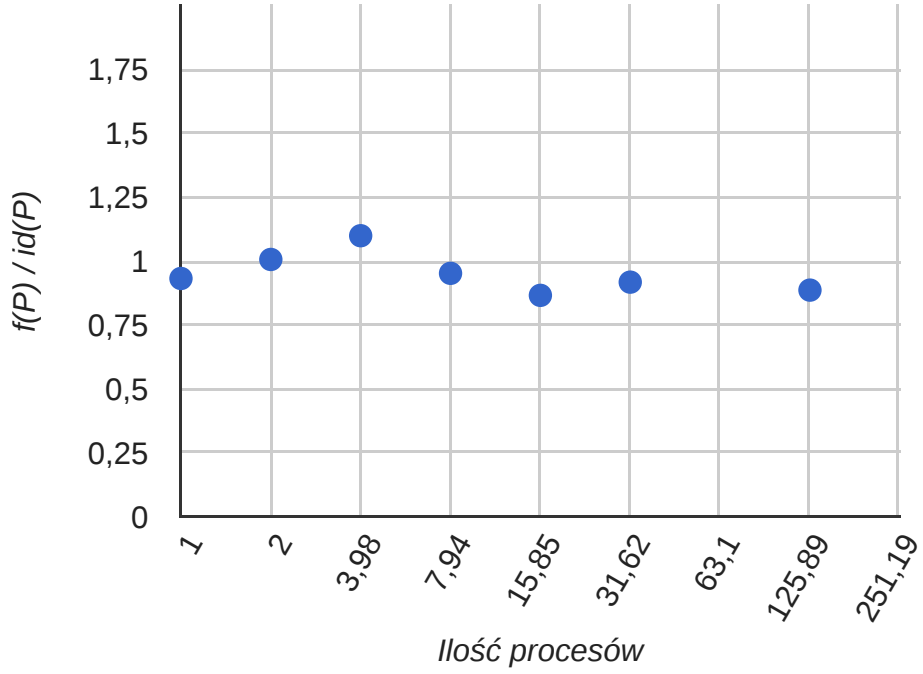
Efektywność obliczeń : N = 25000



Efektywność obliczeń : N = 30000



Efektywność obliczeń : N = 40000

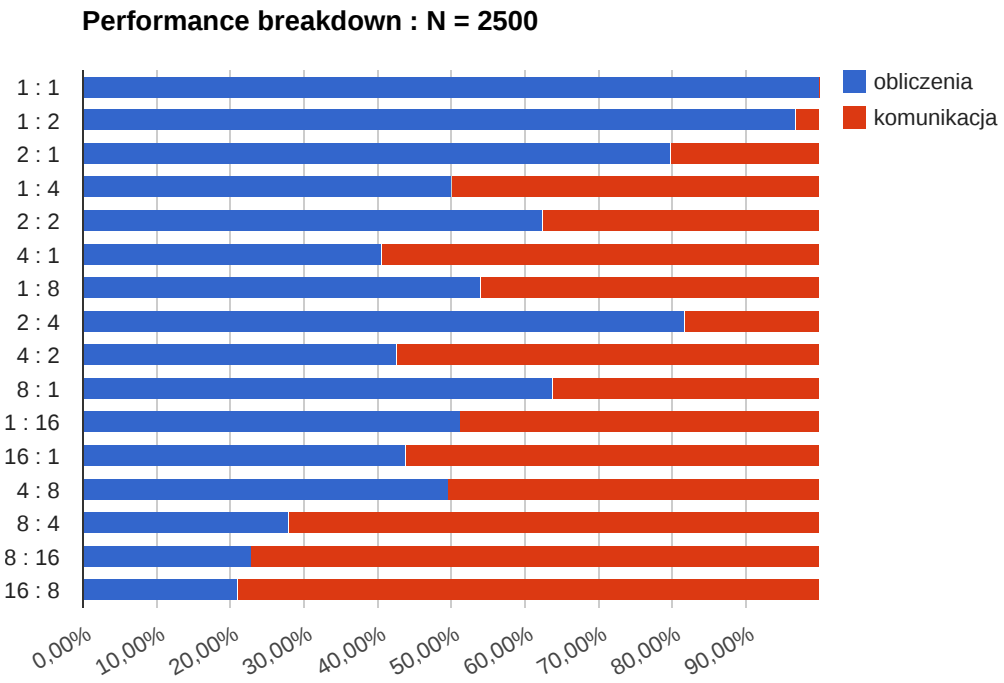


3.6.1 Performance breakdown

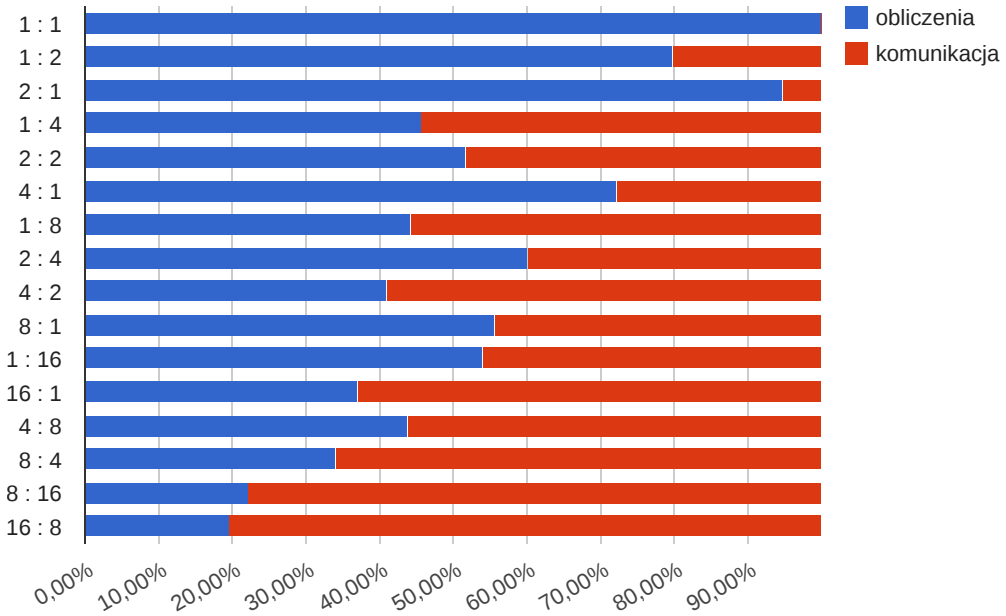
Na koniec pozostał performance breakdown, czyli podział czasu ze względu na typ wykonywanych czynności. Poniższe wykresu ukazują podział całkowitego czasu wykonania na obliczenie i komunikację.

Jak widać komunikacja zajmuje dużą, a od pewnego momentu wręcz większą część czasu. Udział ten wydaje się być słabo skorelowany z rozmiarem siatki, za to mocno z ilością procesów.

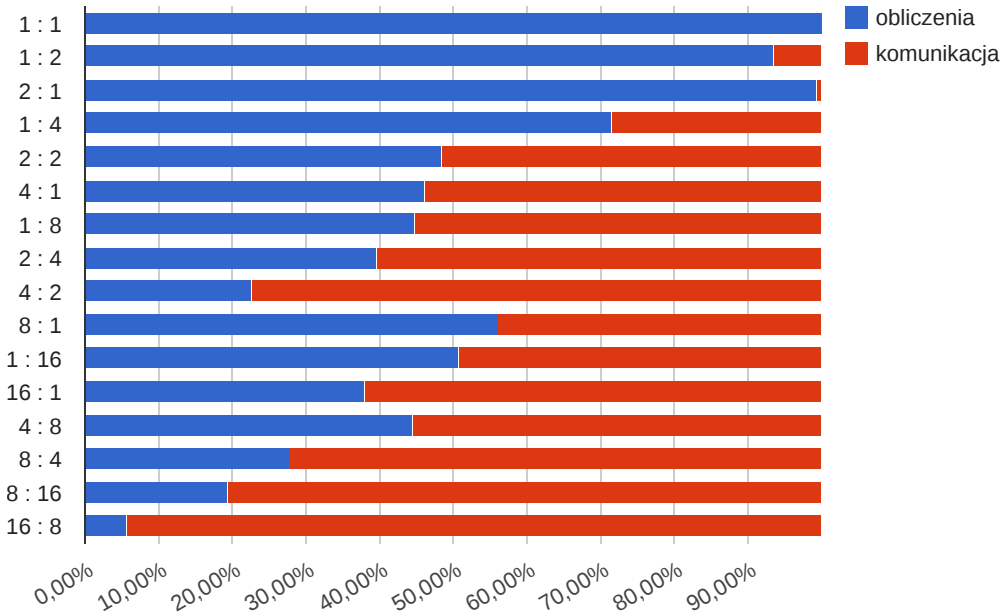
Ponadto, konfiguracje z większą ilością rdzeni niż maszyn ponownie wydają się mieć pewną przewagę pod względem części czasu przeznaczonego na obliczenia.



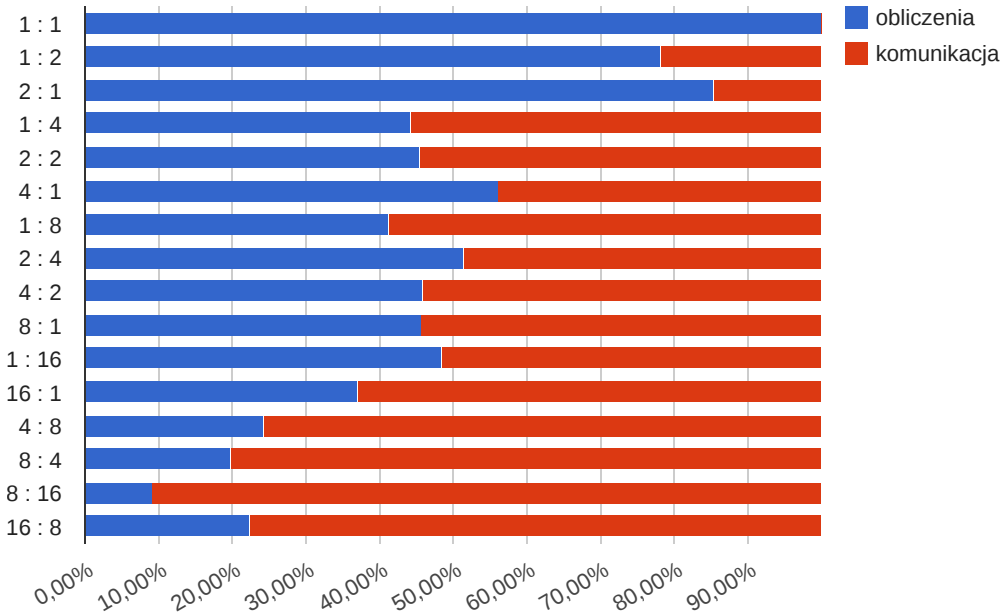
Performance breakdown : N = 5000



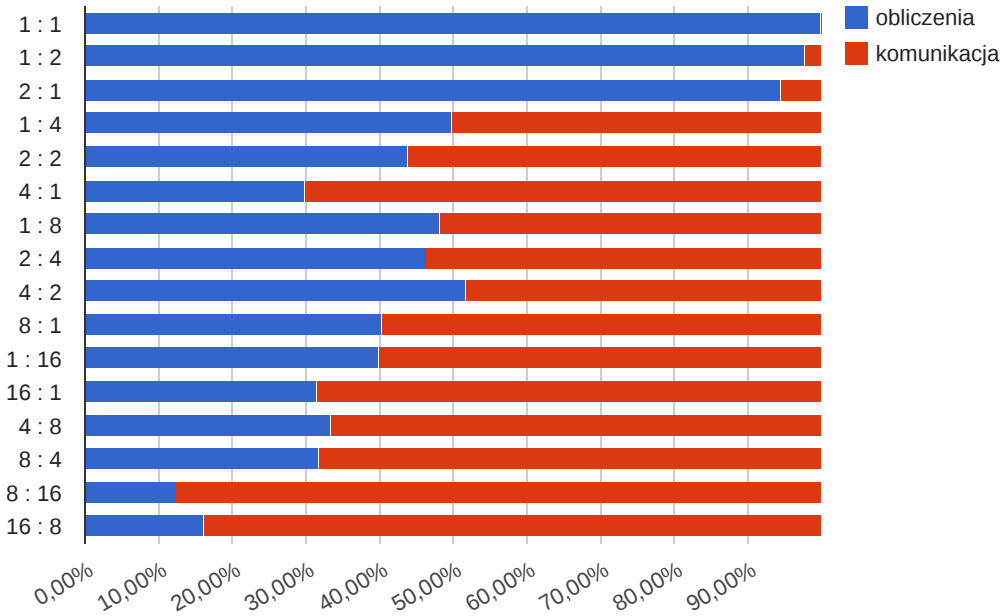
Performance breakdown : N = 10000



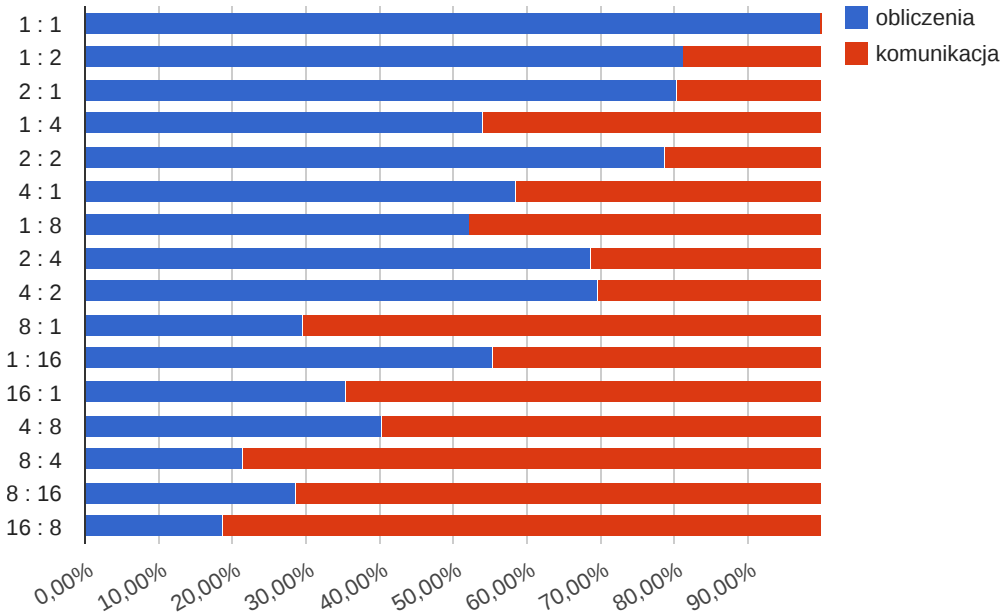
Performance breakdown : N = 15000



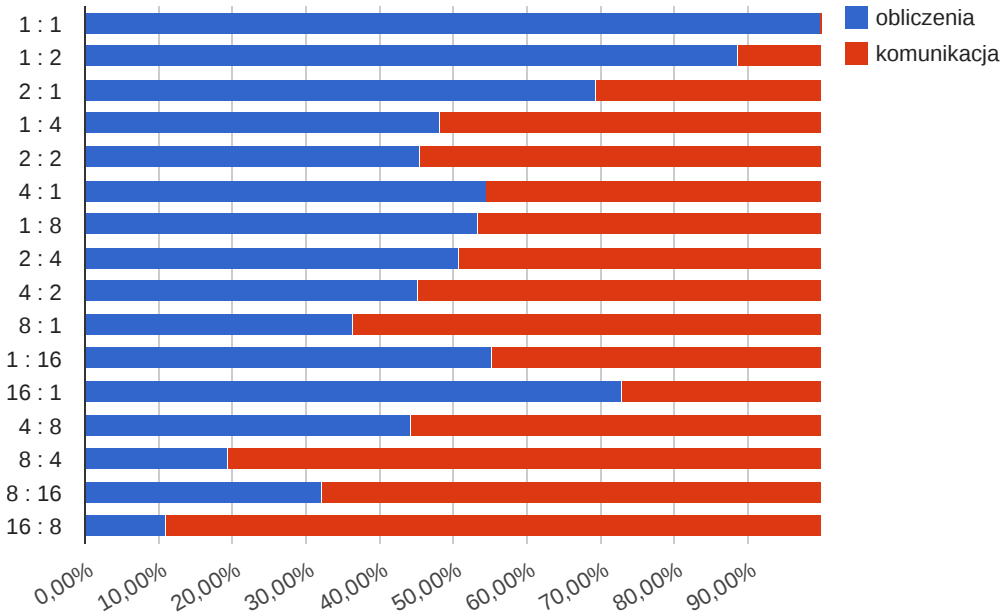
Performance breakdown : N = 20000

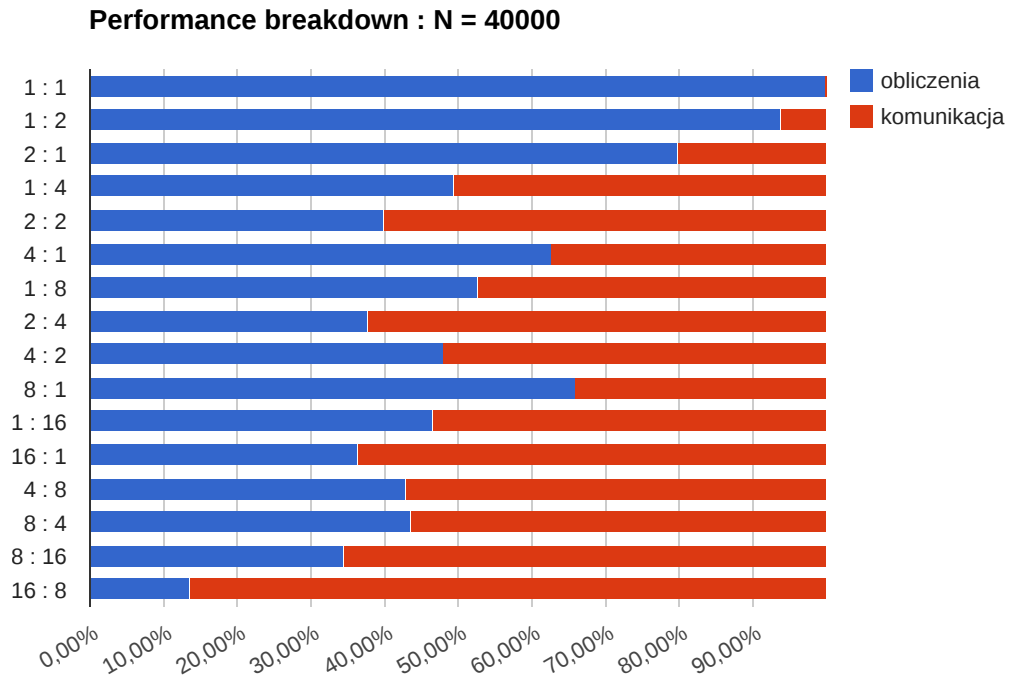


Performance breakdown : N = 25000



Performance breakdown : N = 30000





4 Podsumowanie

Program współbieżny skaluje się, chociaż nie tak dobrze jak można by tego oczekiwać / chcieć. Wina, wyraźnie leży po stronie komunikacji i synchronizacji między procesami. Można by spróbować to naprawić zmieniając sposób dzielenia siatki między procesy (tak, by zmniejszyć ilość przesyłanych danych) oraz zwiększając asynchroniczność komunikacji (np. podczas ustalania maksymalnej różnicy wartości punktów).

Ponadto, warto by przeanalizować program pod względem trafień w pamięć podręczną. Być może niewielka zmiana struktur danych bądź kolejności wykonywania obliczeń mogła by spowodować znaczne przyspieszenie.