Politechnika Krakowska

Wydział inżynierii Elektrycznej i Komputerowej

Studia Niestacjonarne,Infotronika

Sprawozdanie z przedmiotu:

Programowanie Równoległe

Prowadzący: mgr inż. Sławomir Bąk

Temat:

Raport z projektu

Bąbelkowe sortowanie parzyste-nieparzyste

Wykonali:
Szymon Stuglik
Kuba Gwiazda

1. Algorytm

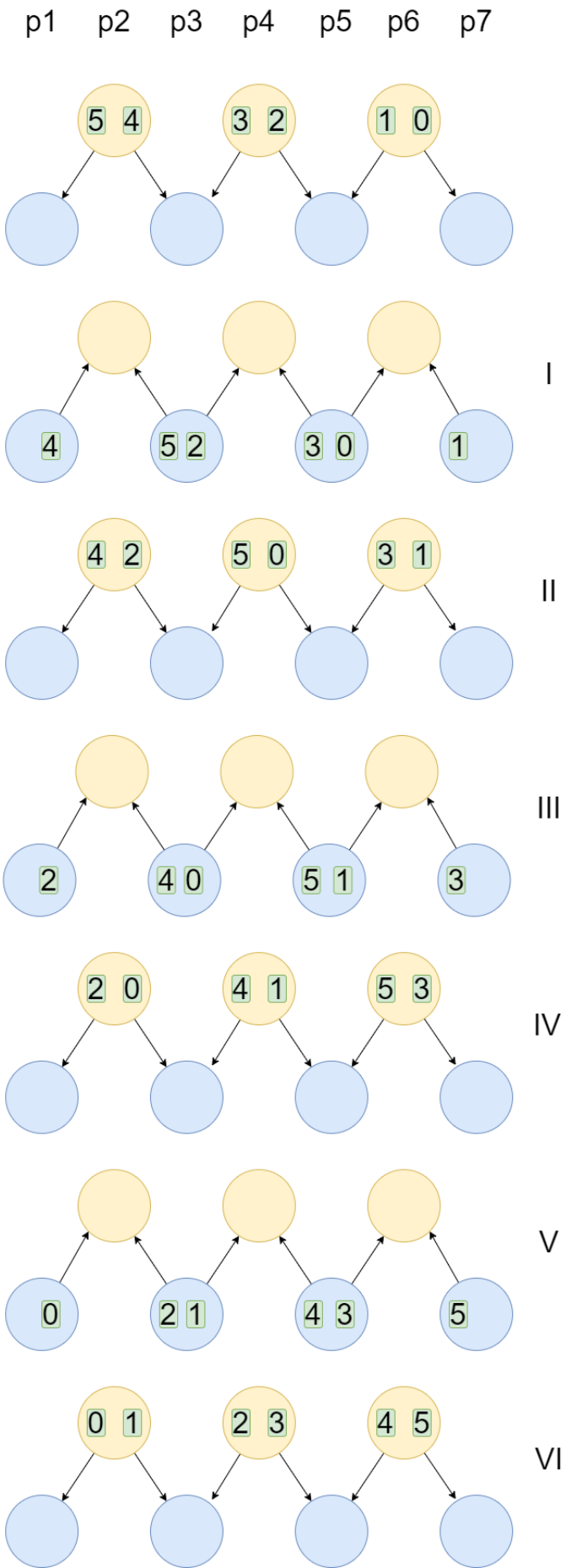
Zadane sortowanie oscylacyjne po dogłebnej analizie okazało się dwu fazowym sortowaniem bąbelkowym. Fakt ten najprościej zaobserwować wykonując przykladowe sortowanie.

Algorytm oscylacyjny

Na starcie sortowana alokowanych jest

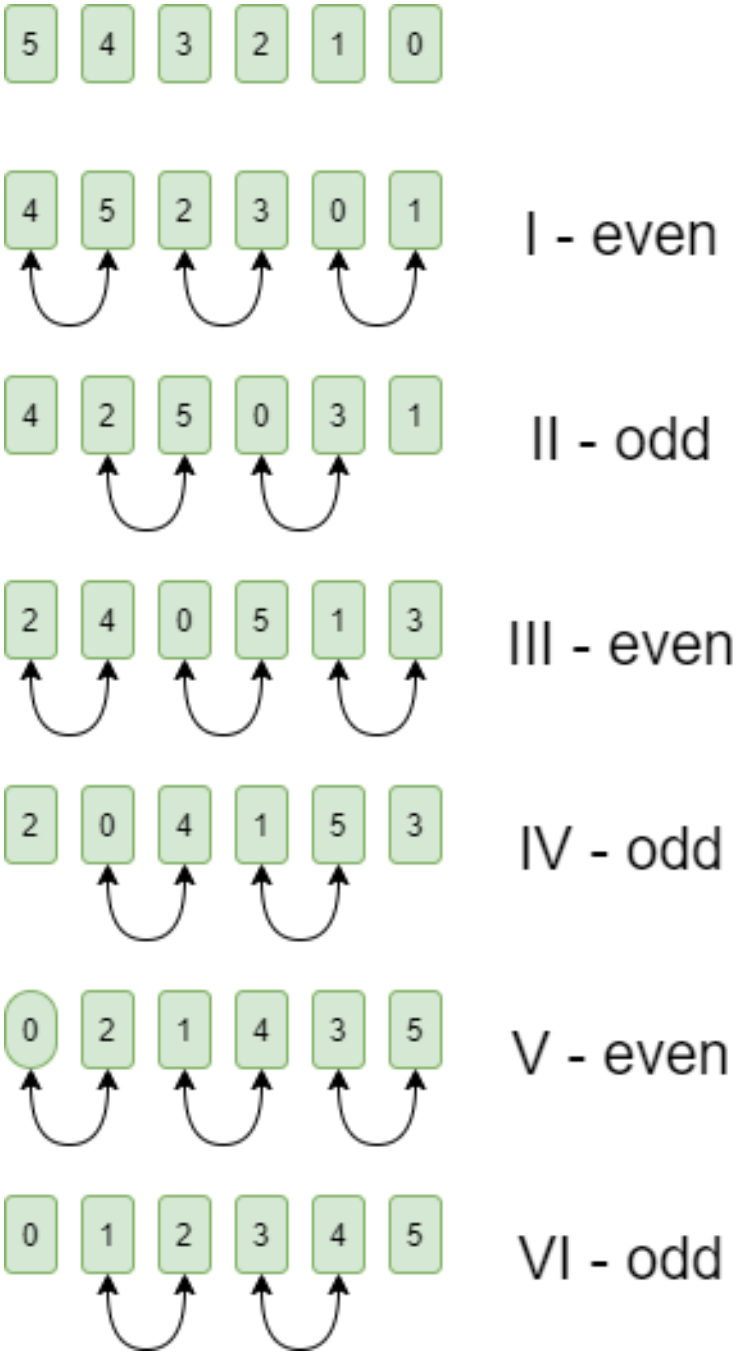
n + 1 procesów, gdzie każdy przy otrzymaniu dwóch wartości przekazuje

mniejszą do lewego procesu i większą do prawego.



Algorytm bąbelkowy

Na starcie programu alokowanych jest n / 2 + 1 procesów, którą otrzymują i zwracają posortowane już liczby do procesu który zlecił porównanie.



Podsumowanie

Jak widać fazy obydwu algorytmów kończą się tym samem układem sortowanej tablicy.

W sortowaniu oscylacyjnym jednak w każde fazie 3 lub 4 procesy nic nie robią. W wersji bąbelkowej w co drugiej fazie tylko jeden proces jest marnowany.

2. Program

Do wykonaniu programu równoległe posłużyliśmy się standartem MPI, wykorzystującym przesyłanie wiadomości.

Wybraliśmy implementacje MPICH, program napisany jest w c++17.

Program generuje losową nieposortowaną tablicę elementów w procesie głównym.

Procesy robocze oczekuja na żądanie porównania dwóch liczb, lub żadanie zakończenia procesu.

Głowny proces również zajmuje się interpretacją argumentów, logowaniem, debugowaniem i wyświetleniem statystyk.

Prosta natura procesów roboczych z góry sprawia że obsługa przesyłania wiadomości bedzie trwała dłużej niż samo porównanie, dodane zostało opóźnienie porównania symulujace trudne obliczenia.

Aby program był bardziej elastyczny i mniej zależny od ilości otwartych procesów, wykonywanie faz może zostać podzielone tak aby jak największa ilośc procesów wykonywała obliczenia.

W przeciwnym wypadku rozmiar tablicy dyktowałby konieczną ilośc procesów potrzebną do wykonania zadania.

Wadą jest to że zależnie od ilości procesów i rozmiaru tablicy zasoby mogą nie byc wykorzystane w stu procentach.

Statystyki:

czas zawsze w milisekundach.

- liczba porównań - ile porównań zostało wykonanych.
- Approx single process time - ile czasu trwałoby wykonanie programu jednym wątkiem.
- Approx multi process time - ile czasu trwałoby wykonanie programu wieloma wątkami, ignorując czas przesyłania wiadomości.
- Sorting time - ile czasu trwało faktyczne sortowanie
- Multiprocess overhead time - ile czasu program spędził na wymianie komunikatów i synchronizacji procesów.

Statystyki

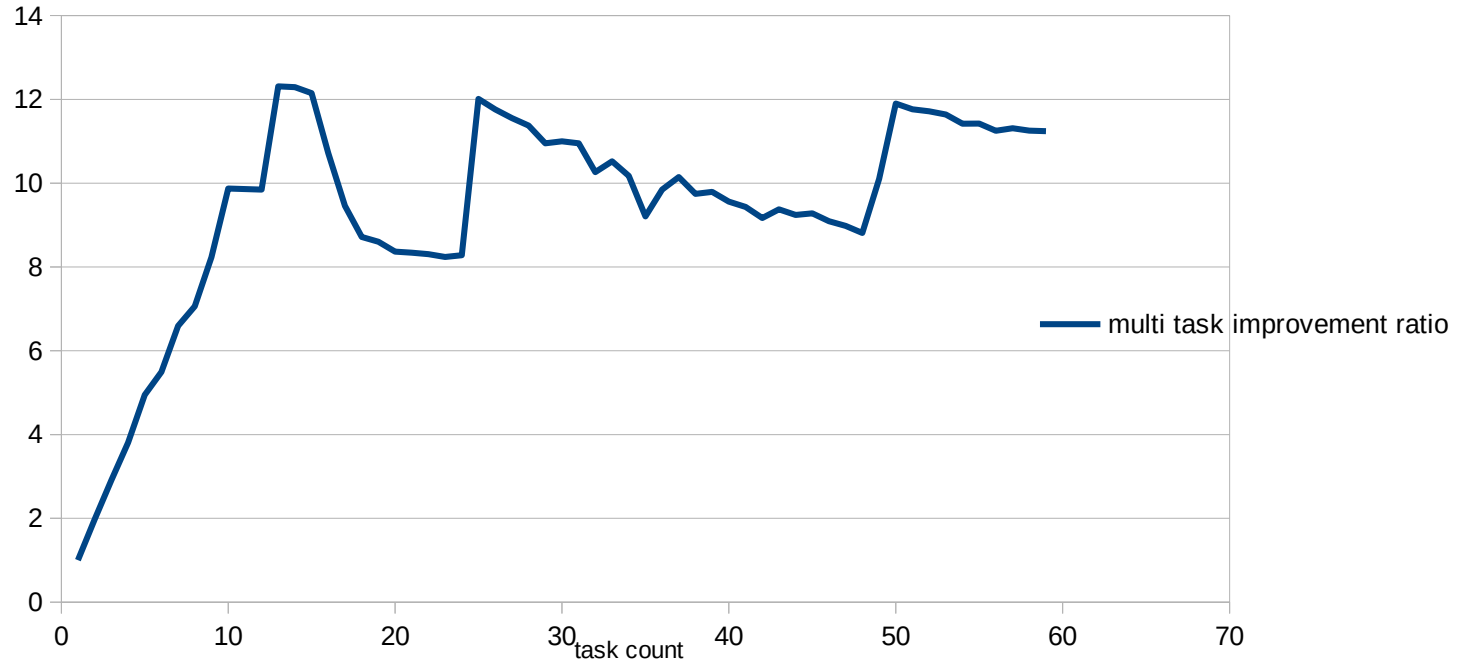
Faktyczna liczba dostępnych procesów to 16.

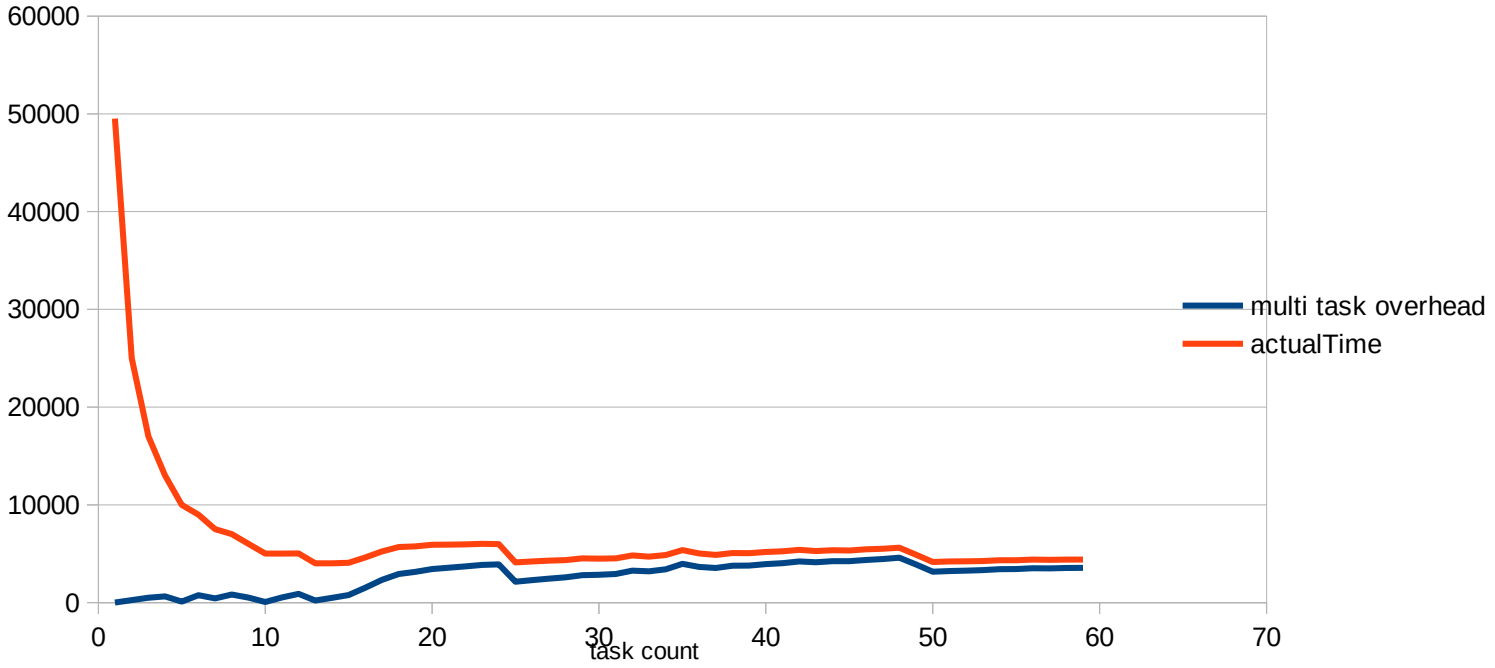
Program został uruchomiony na windows 10 za pomocą WSL'a.

Tablica 100 elementów opóźnienie porównania 10 ms.

4950 porównań.

multi task improvement ratio - o ile szybciej program został wykonany wieloprocesowo w porównaniu do jednego procesu.

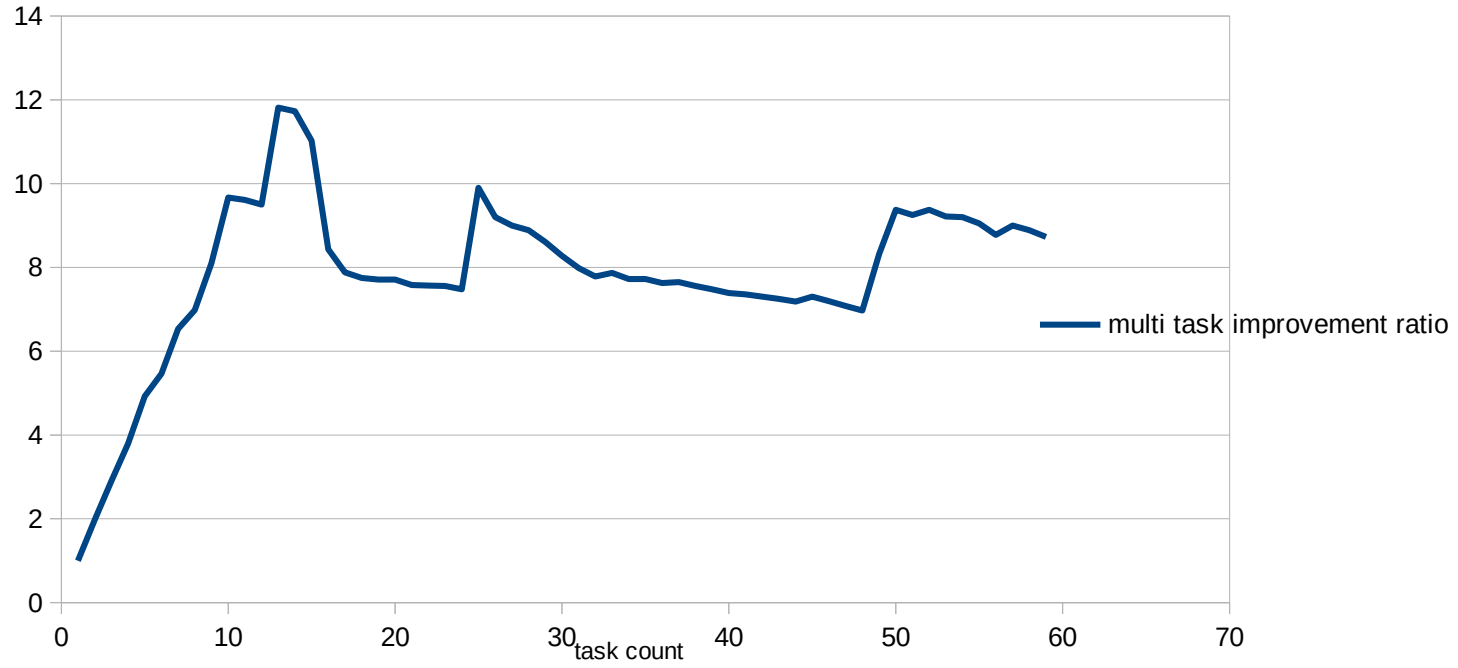


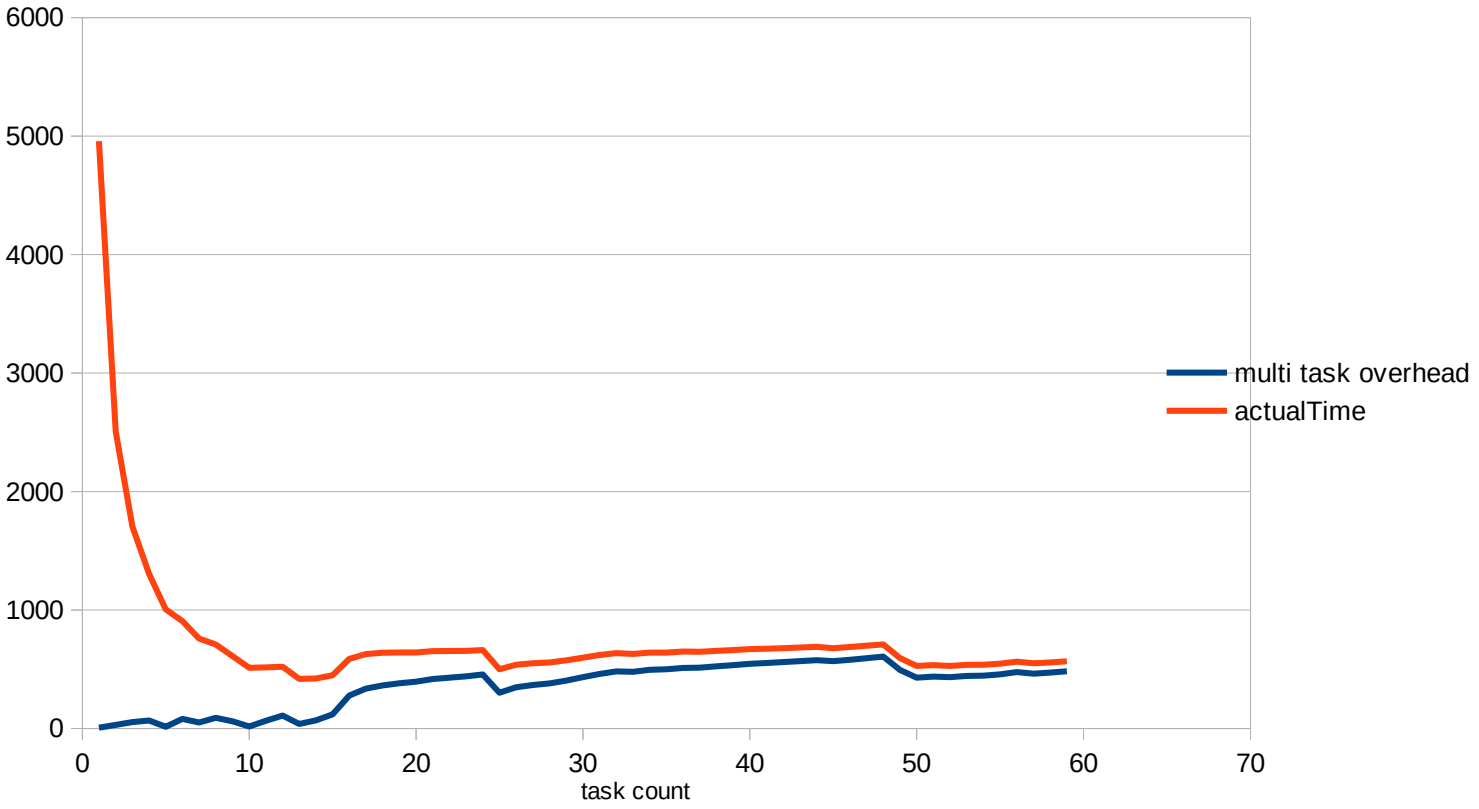


Tablica 100 elementów opóźnienie porównania 1 ms.

4950 porównań.

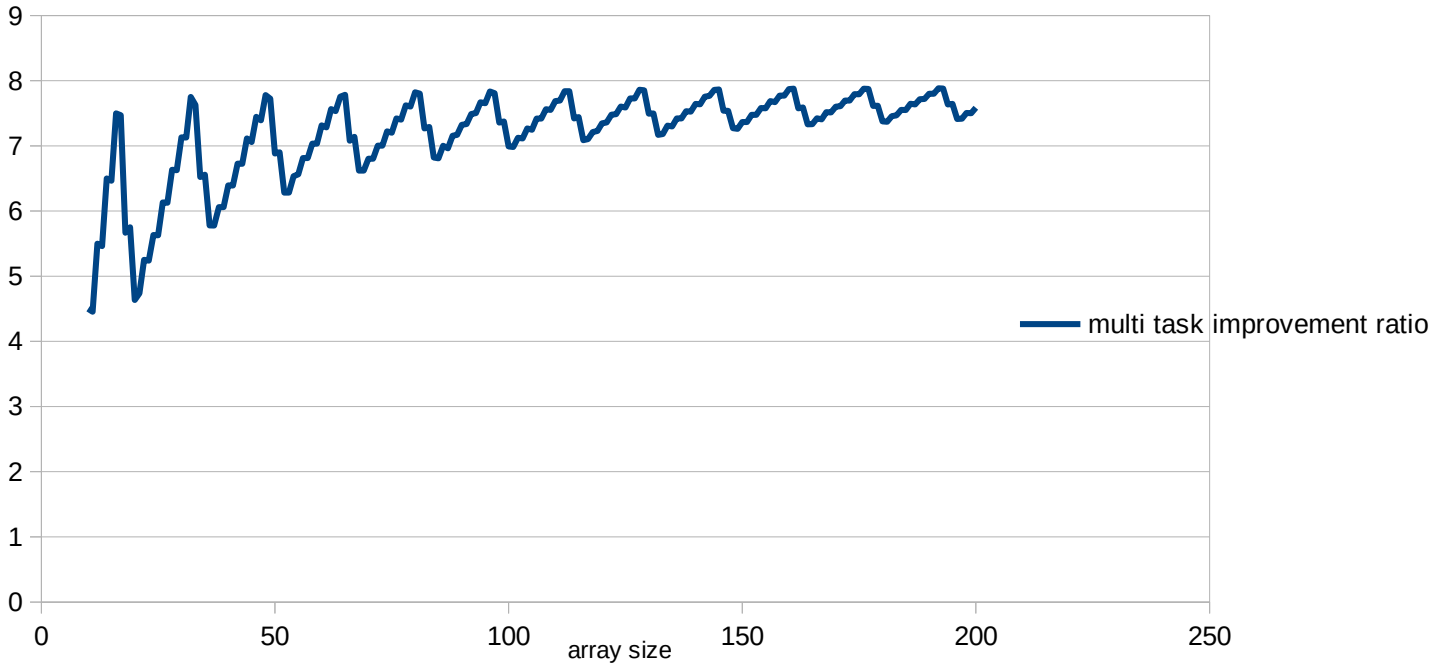
multi task improvement ratio - o ile szybciej program został wykonany wieloprocesowow porównaniu do jednego procesu.



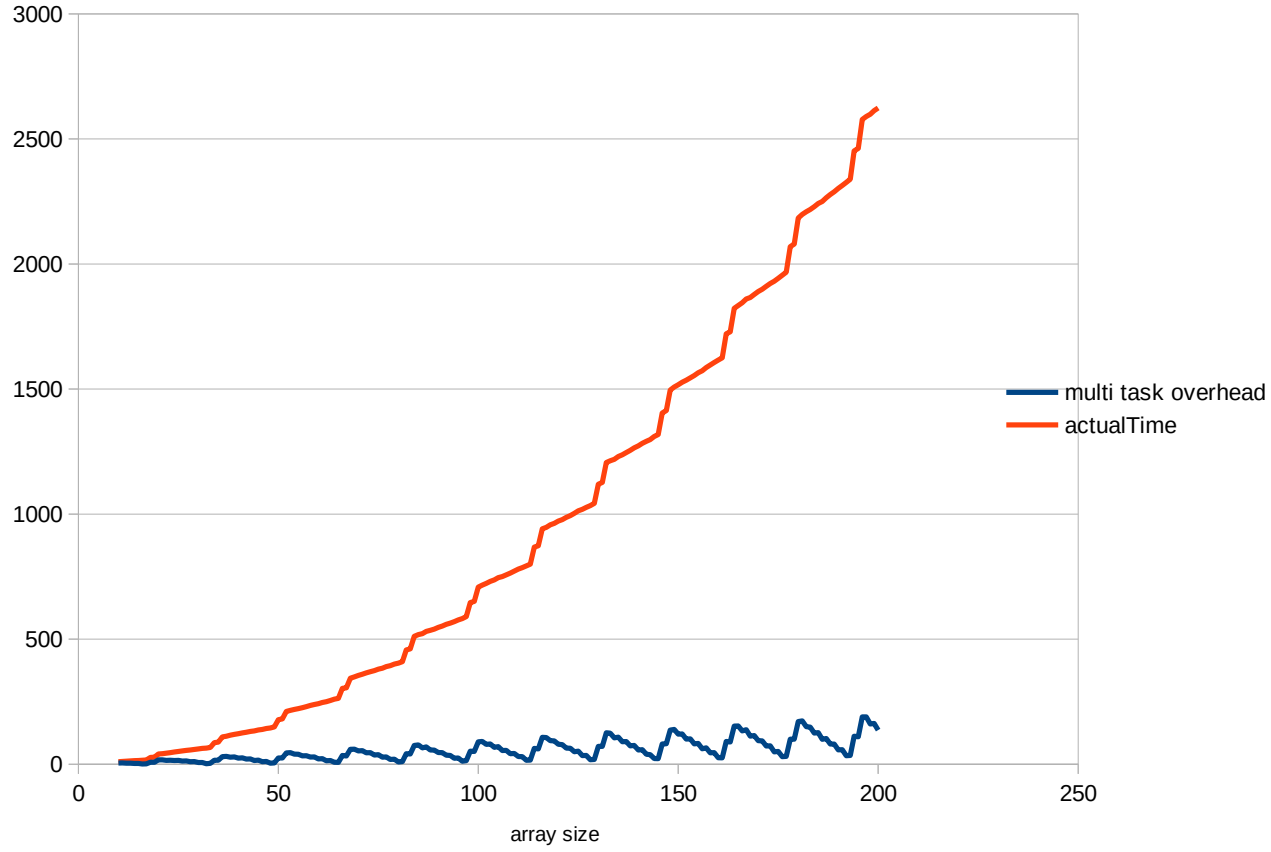


8 procesów, opóźnienie 1 ms.

Ten wykres demonstruje jak algorytm zachowuje w zależności czy liczba procesówdobrze odpowiada rozmiarowi tablicy, czasem dodanie nowego procesu tworzy sytuacje gdzie proces ze względu na podzielnośc tablicy może nie być wykorzystany w stu procentach.



Tutaj widać złożonośc obliczeniową algorytmu $O(n^2)$:



Wnioski

Programy równoległe potrafią wielokrotnie zwiększyć szybkość działania algorytmu, jednak wymaga to odpowienich zasobów sprzętowych. Próby zwiększania "na siłe" liczy procesów nie dają owocnych rezultatów. Synchronizacja wątków i wymiana komunikatów zajmuje coraz więcej czasu.

Gdy sama operacja wykonywana przez procesu robocze jest krótsza niż czas komunikacji z tym procesem, program równoległy jest z góry skazany na nie powodzenie.

Najlepszym zastosowaniem programów równoległych są intesywne obliczeniowo łatwo podzielne algorytmy, gdzie czas komunikacji bedzie zajmował jak najmiejszą część czasu faktycznego wykonania programu.