Wydział Elektrotechniki, Automatyki i Informatyki

Politechnika Opolska

AUTOREFERAT ROZPRAWY DOKTORSKIEJ

Analiza efektywności równoległych implementacji wybranych algorytmów optymalizacji statycznej.

mgr inż. Marek Machaczek

Promotor: dr hab. inż. Jan Sadecki

Opole 2023

Spis treści

Rozdział 1 Wprowadzenie w tematykę pracy 3
Rozdział 2 Teza, cele i zakres pracy 6
Rozdział 3 Przykładowe wyniki obliczeń 8
3.1 Algorytm Gaussa-Seidela 10
3.2 Z modyfikowany równoległy algorytm Gaussa-Seidela z modyfikacją bazy (ARm) 14
3.3 Algorytm Powella 16
3.3.1 Algorytm Powella bez modyfikacji bazy 16
3.3.2 Algorytm Powella z modyfikacją bazy 18
3.3.3 Algorytm Powella z przeszukiwaniem wiązką 18
3.4 Algorytm Chazana-Mirankera 20
3.5 Algorytm Neldera-Meada 21
3.6 Porównanie czasów wykonywania wybranych algorytmów 27
Rozdział 4 Optymalizacja dynamiczna 29
Rozdział 5 Uwagi końcowe 36
5.1 Wnioski 36
5.2 Oryginalne wyniki pracy naukowej 44
5.3 Kierunki dalszych badań 45
Bibliografia 46

Rozdział 1

Wprowadzenie w tematykę pracy

W wielu dziedzinach nauki, ekonomii, finansów, przemysłu i innych można spotkać się z różnymi problemami wymagającymi przeprowadzenia optymalizacji [1, 2, 3, 4, 8, 9, 12, 13, 14, 19, 20, 24, 28, 31, 32, 33, 34, 36, 37, 39, 40, 41, 42]. Problemy te mogą być niejednokrotnie bardzo złożone, a przy tym nierzadko niezmiernie wymagające w odniesieniu do nakładu czasu związanego z ich numerycznym rozwiązywaniem. W przedmiotowej literaturze opisanych jest wiele algorytmów optymalizacji, często bardzo różnorodnych tak pod względem stopnia zaawansowania wykorzystywanej teorii jak i złożoności ich numerycznej implementacji [5, 10]. Mnogość tych algorytmów wynika z faktu, że praktycznie nie jest znany na tyle uniwersalny algorytm optymalizacji, który sprawdziłby się w odniesieniu do relatywnie szerokiej klasy problemów optymalizacji spotykanych w teorii i w praktyce. W zależności od napotykanego problemu algorytmy te mogą być lepsze lub gorsze tak pod względem dokładności, warunków zbieżności jak też czasu potrzebnego do wyznaczenia optymalnego rozwiązania. Rozwój komputerów oraz niespotykanie szybki wzrost ich wydajności, z jakim mamy ostatnio do czynienia, stworzyły realne podstawy do tego, aby zmierzyć się z problemem rozwiązywania coraz bardziej złożonych problemów optymalizacji, tak pod względem wymiarowości jak i stopnia ich skomplikowania [30]. Dzisiejsze komputery stacjonarne są wyposażone w procesory wielordzeniowe, a klastry komputerowe dzięki połączonej mocy wielu takich procesorów mogą tworzyć systemy komputerowe (tzw. superkomputery), składające się z setek, tysięcy, a obecnie nawet milionów jednostek obliczeniowych [6, 17, 18, 26]. Jednakże wiele znanych algorytmów optymalizacji ma strukturę wyraźnie sekwencyjną i w swojej pierwotnej wersji mogą one wykorzystywać jedynie część mocy obliczeń współczesnych komputerów, na których byłyby one uruchamiane. Dlatego zasadne wydaje się być zrównoleglenie takich algorytmów w celu wykorzystania jak największej części dostępnej mocy obliczeniowej współczesnych procesorów [11, 23].

Zadanie zrównoleglenia algorytmu o typowej strukturze sekwencyjnej może nie być zadaniem trywialnym. Problem ten może wymagać głębszej analizy danego algorytmu oraz

często znaczącej ingerencji w jego strukturę. Taka ingerencja może mieć jednak negatywne skutki. Na przykład, algorytm równoległy uruchomiony na jednym wątku/procesie (czyli algorytm który będzie wykonywał się sekwencyjnie, lecz według schematu równoległego) może być wolniej zbieżny do rozwiązania optymalnego niż oryginalna jego wersja sekwencyjna. Jednakże pomimo takich lub podobnych wad i trudności, jakie mogą towarzyszyć zrównoleglaniu algorytmu, otrzymany algorytm równoległy ma niezaprzeczalną zaletę, jaką jest możliwość równoległego wykonywania. W zależności od efektywności zrównoleglenia, prowadzi to zazwyczaj do lepszego wykorzystania dostępnej mocy obliczeniowej. Odpowiednio dobrany oraz zrównoleglony algorytm może być wielokrotnie szybszy niż jego tradycyjna wersja.

Zrównoleglanie algorytmów powinno w zasadzie dotyczyć praktycznie każdego elementarnego zadania występującego w algorytmie, nawet niejednokrotnie tak prostego zadania, jakim jest np. badanie spełnienia kryterium stopu. Jednak takie podejście wymaga, aby każde z tych zadań było możliwe do zrównoleglenia i by było to zasadne. Istnieją jednak przypadki, w których zrównoleglenie obliczeń może być niezwykle trudne, lub wręcz niemożliwe do przeprowadzenia, a „zyski" płynące w takich sytuacjach ze zrównoleglenia mogą być nieopłacalne. W systemach klastrowych takie podejście, w porównaniu do systemów z pamięcią współdzieloną, może być jeszcze mniej opłacalne, ponieważ dla niektórych zadań obliczeniowych (nawet o krótkim czasie realizacji) może być potrzebna dodatkowa komunikacja i/lub synchronizacja pomiędzy jednostkami obliczeniowymi. Może to być przyczyną wzrostu sumarycznego czasu związanego z równoległą realizacją obliczeń. W końcowym rozrachunku ten dodatkowy narzut na komunikację może zniwelować „zyski" płynące ze zrównoleglania każdego, nawet najmniejszego, zadania. Na ogólną opłacalność zrównoleglenia zadania mają wpływ dostępne zasoby, odpowiedni podział i przechowywanie danych, które mogą zmniejszyć narzut na komunikację oraz odpowiednie dopasowanie do architektury równoległej, na której ma być przetwarzany dany algorytm. Generalnie najlepszy „zysk", pod względem czasu trwania obliczeń daje z reguły zrównoleglenie najbardziej masywnych zadań występujących w algorytmie.

Równoległe wersje algorytmów optymalizacji mogą mieć zastosowanie wszędzie tam gdzie istotną rolę pełni czas trwania obliczeń i dostępna jest niewykorzystana moc obliczeniowa. Obecnie większość, o ile nie wszystkie, komputery posiadają procesory wielordzeniowe. W dużym uproszczeniu każdy rdzeń działa jak osobny procesor (rdzenie mają dostęp do pamięci współdzielonej, a w przypadku procesorów zazwyczaj mówi się o pamięci rozproszonej

(klastry)). W porównaniu z architekturą jednordzeniową architektura wielordzeniowa/wieloprocesorowa pozwala na zwielokrotnienie mocy obliczeniowej, a odpowiednie jej wykorzystanie może znacząco wpłynąć na czas wykonywania obliczeń.

Zrównoleglenie algorytmów może prowadzić również do poszerzenia klasy problemów, do której stosowany był niezrównoleglony algorytm, poprzez możliwe, niejednokrotnie znaczne, skrócenie czasu wykonywania obliczeń [35]. Jednym z ograniczeń sekwencyjnych algorytmów optymalizacji jest czas wykonywania obliczeń, który może być z punktu widzenia niektórych zastosowań nieakceptowalny. Zastosowanie zrównoleglenia względem takiego algorytmu może wielokrotnie skrócić czas obliczeń, a co z tym idzie algorytm może zostać wykorzystany w bardziej czasochłonnych przypadkach.

Jak pokazuje analiza literatury przedmiotu, rozwój metod optymalizacji w kontekście równoległego ich przetwarzania, w porównaniu do innych dziedzin obliczeń numerycznych, wydaje się jeszcze stosunkowo niewystarczający. Pomimo rozwoju komputerów często w dalszym ciągu wykorzystuje się w aplikacjach klasyczne sekwencyjne metody optymalizacji, dlatego badania prowadzone w kierunku zrównoleglania algorytmów optymalizacji są aktualnie nadal potrzebne i uzasadnione.

Rozdział 2

Teza, cele i zakres pracy

Rozważając możliwość i celowość równoległej implementacji algorytmów optymalizacji statycznej sformułowano tezę:

„Przetwarzanie równoległe stwarza realne możliwości znaczącej poprawy efektywności realizacji analizowanych w pracy metod i algorytmów optymalizacji statycznej poprzez konstrukcję nowych rozwiązań algorytmicznych bardziej dostosowanych do wymogów architektury równoległej oraz poprzez poszukiwanie efektywnych implementacji tych algorytmów uwzględniających specyfikę architektury równoległej klastrów obliczeniowych, procesorów wielordzeniowych oraz procesorów graficznych GPGPU."

Następujące cele pracy zostały przedstawione w celu potwierdzenia powyższej tezy:

1. Opracowanie równoległych realizacji wybranych algorytmów optymalizacji statycznej dla systemów równoległych takich jak: klastry obliczeniowe, procesory wielordzeniowe oraz procesory GPGPU, z jednoczesnym poszukiwaniem najbardziej efektywnych implementacji badanych algorytmów w wykorzystanych systemach.
2. Analiza możliwości, sposobów oraz efektywności rozwiązywania złożonych (wielowymiarowych) zagadnień optymalizacji statycznej w równoległych systemach wieloprocesorowych rozważanych typów.
3. Poszukiwanie nowych konstrukcji i rozwiązań algorytmicznych konkurencyjnych pod względem efektywności w stosunku do standardowych algorytmów.
4. Zastosowania badanych algorytmów do rozwiązania przykładowych wielowymiarowych problemów optymalizacji statycznej.

Zakres pracy został przedstawiony poniżej:

1. Analiza teoretyczna w zakresie badania własności konstruowanych algorytmów optymalizacji.
2. Równoległa implementacja rozważanych metod optymalizacji statycznej.

3. Projektowanie oraz implementacja wybranych algorytmów optymalizacji statycznej z uwzględnieniem różnych modeli przetwarzania równoległego tak w zakresie oprogramowania jak i sprzętu komputerowego.
4. Tworzenie teoretycznych modeli pozwalających na ocenę efektywności realizacji algorytmów równoległych.

Początkowa część pracy ma charakter wprowadzający w zagadnienia omawiane w pracy. Zawarto w niej ogólne wprowadzenie w tematykę pracy, przegląd literatury dotyczącej poruszanych problemów w rozprawie oraz przedstawiono tezę, cele i zakres pracy. Przedstawiono również wprowadzenie w problematykę optymalizacji, sformułowanie problemu optymalizacji i klasyfikację algorytmów optymalizacji oraz ich własności.

Główna część zawiera przegląd bezgradientowych algorytmów optymalizacji statycznej w ich oryginalnych wersjach oraz proponowanych równoległych implementacji, które wykorzystują architekturę równoległą. Opisano także przykładowe funkcje testowe. Sformułowano również przykładowe problemy optymalizacji dynamicznej, których rozwiązywanie mogłoby zostać oparte o równoległe algorytmy optymalizacji statycznej. Przedstawiono środowiska testowe, na których przeprowadzono obliczenia, jak również zawarto rezultaty badań poszczególnych algorytmów w zależności od użytej biblioteki wraz z opisami wyników oraz metodologią badań.

Ostatnia część jest podsumowaniem rozprawy. Zawiera wnioski z otrzymanych wyników oraz opis oryginalnych wyników, a także możliwe dalsze kierunki badań dotyczące problematyki zrównoleglania algorytmów optymalizacji.

Dodatek A, zawarty w pracy, przeznaczono na wprowadzenie zagadnień programowania równoległego oraz opis wykorzystanych bibliotek. Zawierają między innymi krótką historię danej biblioteki, opis działania, opis przykładowych funkcji wraz z przykładami ich użycia, sposoby kompilacji i uruchomienia.

Rozdział 3

Przykładowe wyniki obliczeń

Obliczenia przeprowadzono z wykorzystaniem sekwencyjnych oraz równoległych wersji algorytmów optymalizacji statycznej przedstawionych w tej pracy. Badania przeprowadzono dla dużych wymiarów problemu – zazwyczaj od 50 do 1000 z krokiem 50 (w niektórych przypadkach ten zakres był inny).

Wszystkie wyniki zostały przedstawione na wykresach przedstawiających:

- liczbę iteracji potrzebną do osiągnięcia minimum przy spełnieniu założonego kryterium stopu
- porównanie czasów wykonywania algorytmów (w sekundach)
- przyśpieszenie jako współczynnik efektywności zrównoleglenia.

Współczynnik przyśpieszenia został opisany następującym wzorem:

$$S(P,n) = \frac{t_{SEQ}(1,n)}{t_{PAR}(P,n)}$$

gdzie:

- $t_{SEQ}(1,n)$ - czas realizacji algorytmu sekwencyjnego lub w szczególnych przypadkach algorytmu równoległego realizowanego sekwencyjnie na jednym wątku albo na jednym procesie,
- $t_{PAR}(P,n)$ - czas realizacji algorytmu równoległego przy wykorzystaniu P wątków lub P procesów.

Czas $t_{SEQ}(1,n)$ przyjmowano jako czas algorytmu sekwencyjnego, jeżeli zmiany w strukturze algorytmu były nieznaczące. Natomiast jeżeli wprowadzono istotne zmiany algorytmu, względem oryginalnej wersji, przyjmowano czas algorytmu równoległego uruchomionego na jednym wątku/procesie to znaczy algorytm wykonywany był sekwencyjnie, ale według schematu algorytmu równoległego.

Badania w pracy zostały przeprowadzone dla trzech wybranych funkcji testowych:

- funkcji drugiego stopnia

$$\min_{\boldsymbol{x}\in R} f(\boldsymbol{x}) = \sum_{i=1}^{n} (2i - 1)(x_i - (2 + i))^2 + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (x_i - x_j + (j - i))^2 \tag{3.1}$$

- funkcji czwartego stopnia

$$\min_{\boldsymbol{x}\in R} f(\boldsymbol{x}) = \sum_{i=1}^{n} (2i - 1)(x_i - (2 + i))^4 + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (x_i - x_j + (j - i))^4 \tag{3.2}$$

- funkcji Rosenbrocka

$$\min_{\boldsymbol{x}\in R} f(\boldsymbol{x}) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \tag{3.3}$$

Wszystkie te funkcje miały znane minima globalne.

Jako punkt początkowy przyjmowano początek układu współrzędnych (0,0) dla wszystkich algorytmów oprócz metody Neledra-Meada.

Obliczenia przeprowadzono na kilku różnych platformach. Główną platformą pomiarową był klaster znajdujący się w Instytucie Informatyki Politechniki Opolskiej. Do przeprowadzenia obliczeń wykorzystano serwer o następującej konfiguracji:

- 2 x Intel Xeon 2.3 GHz (łącznie 20 rdzeni)
- 128 GB pamięci RAM
- dysk SSD (odczyt 2000 MB/s oraz zapis 1000 MB/s)
- system operacyjny typu Linux.

Niektóre obliczenia (np. algorytm Neldera-Meada) ze względów technicznych przeprowadzono na 8 rdzeniowym komputerze o konfiguracji:

- AMD RYZEN 7 5800X 3.8 GHz (8 rdzeni)
- 32 GB pamięci RAM

- dysk SSD
- system operacyjny typu Linux.

Część obliczeń, szczególnie powiązane z CUDA, zostały wykonane na komputerze o specyfikacji:

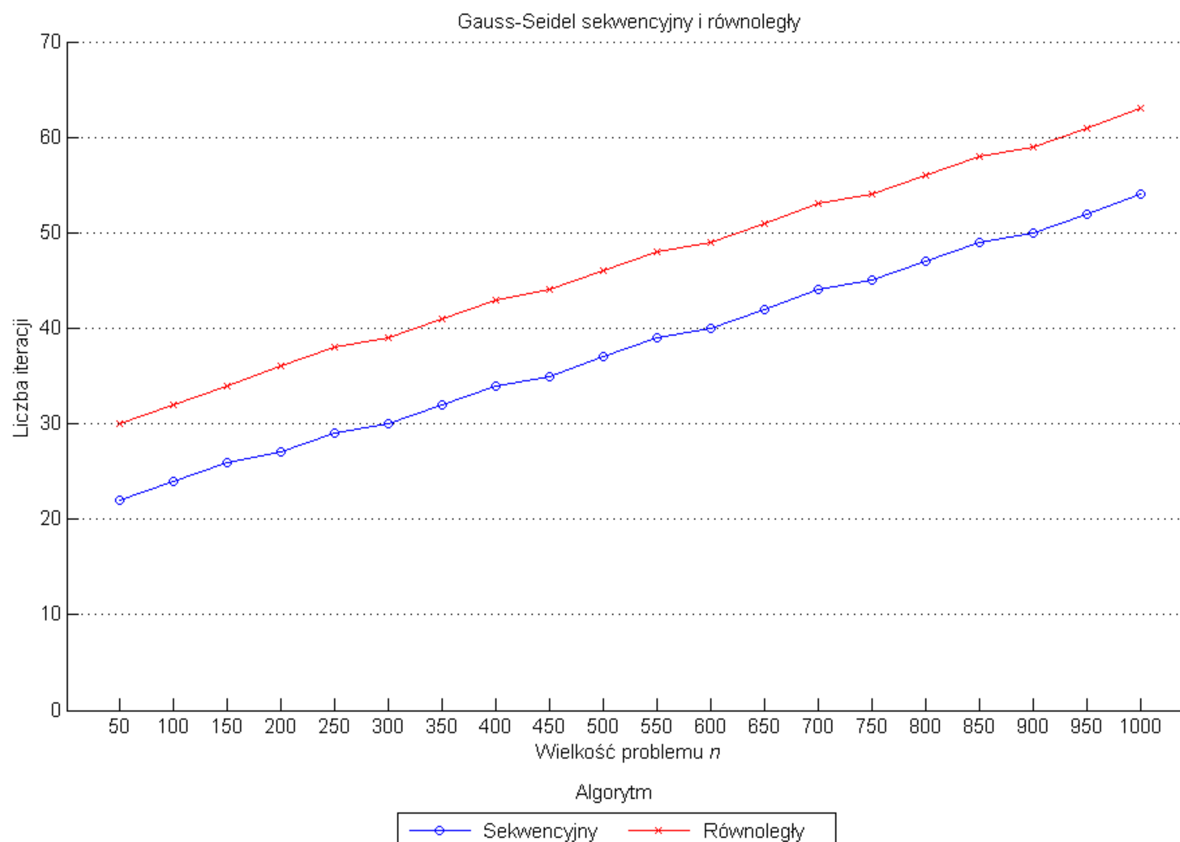
- Intel Core i5-3570K
- GeForce GTX660.

Obliczenia realizowano przy wykorzystaniu bibliotek: OpenMP, MPI oraz CUDA. Ze względu na to, że oba wielordzeniowe procesory uczelnianego klastra, na których przeprowadzono obliczenia, fizycznie znajdowały się na jednej płycie głównej (współdzielona pamięć) i nie było możliwości zmierzenia wpływu przesyłania komunikatów przez sieć postanowiono zaprezentować otrzymane wyniki dla biblioteki OpenMP.

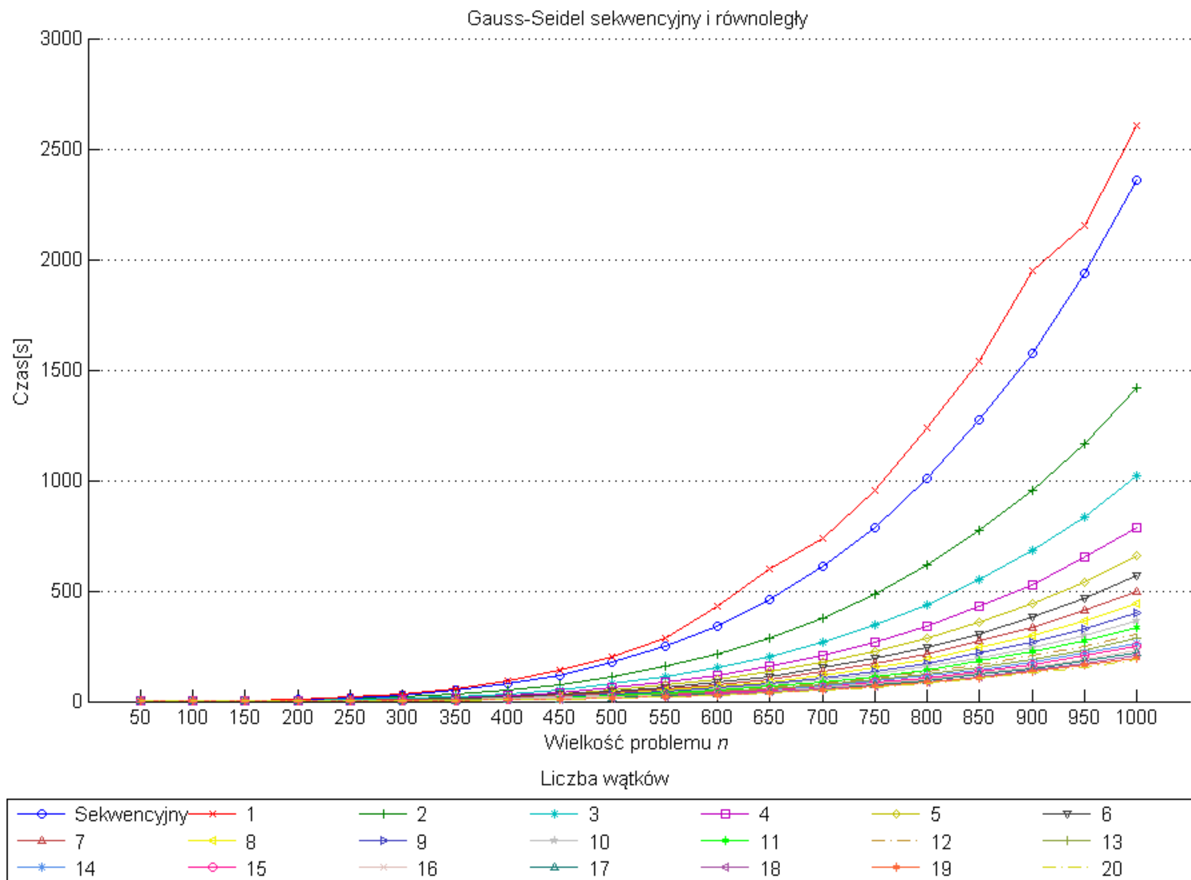
Biblioteki OpenMP, MPI oraz CUDA zostały omówione w dodatku A, w którym omówiono również inne zagadnienia dotyczące równoległości.

3.1 Algorytm Gaussa-Seidela

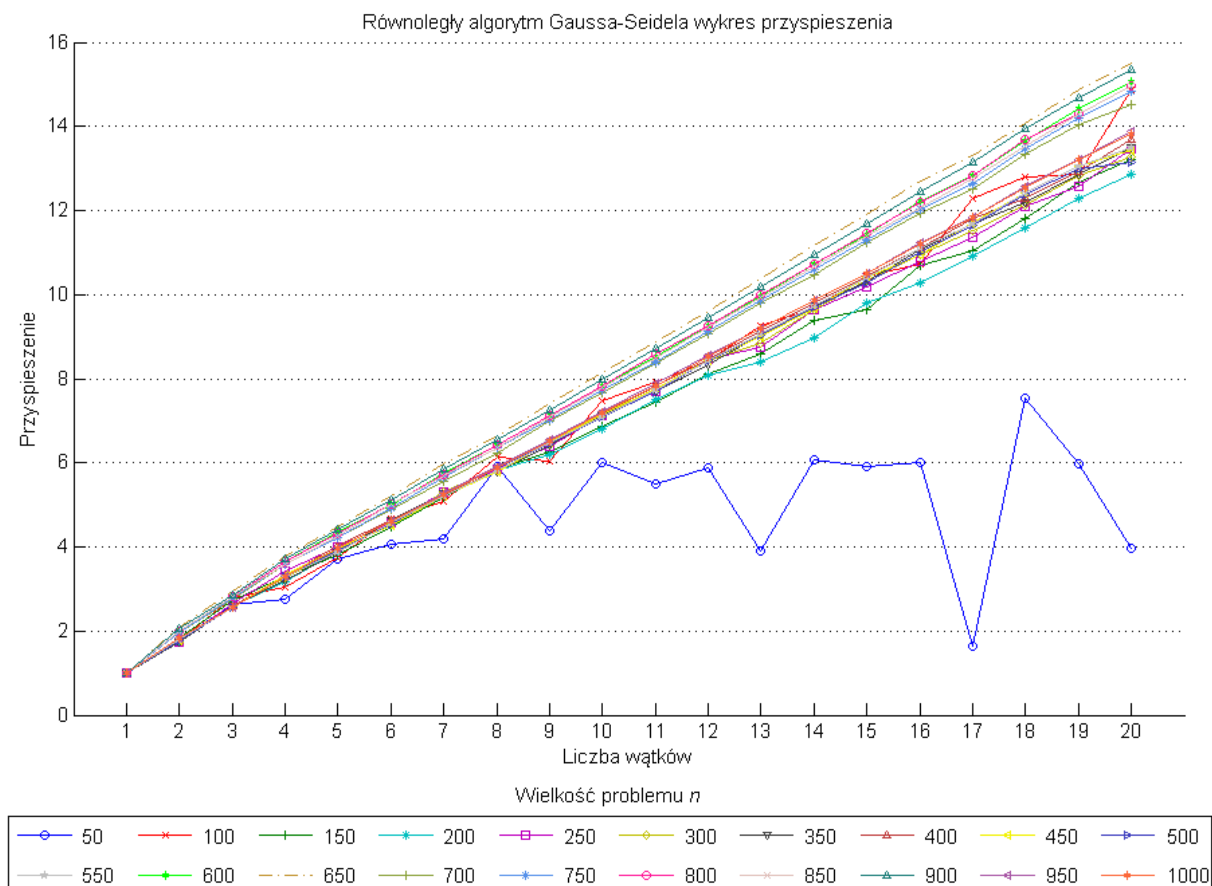
Wyniki dla funkcji 4 stopnia (3.2) przedstawiono poniżej.



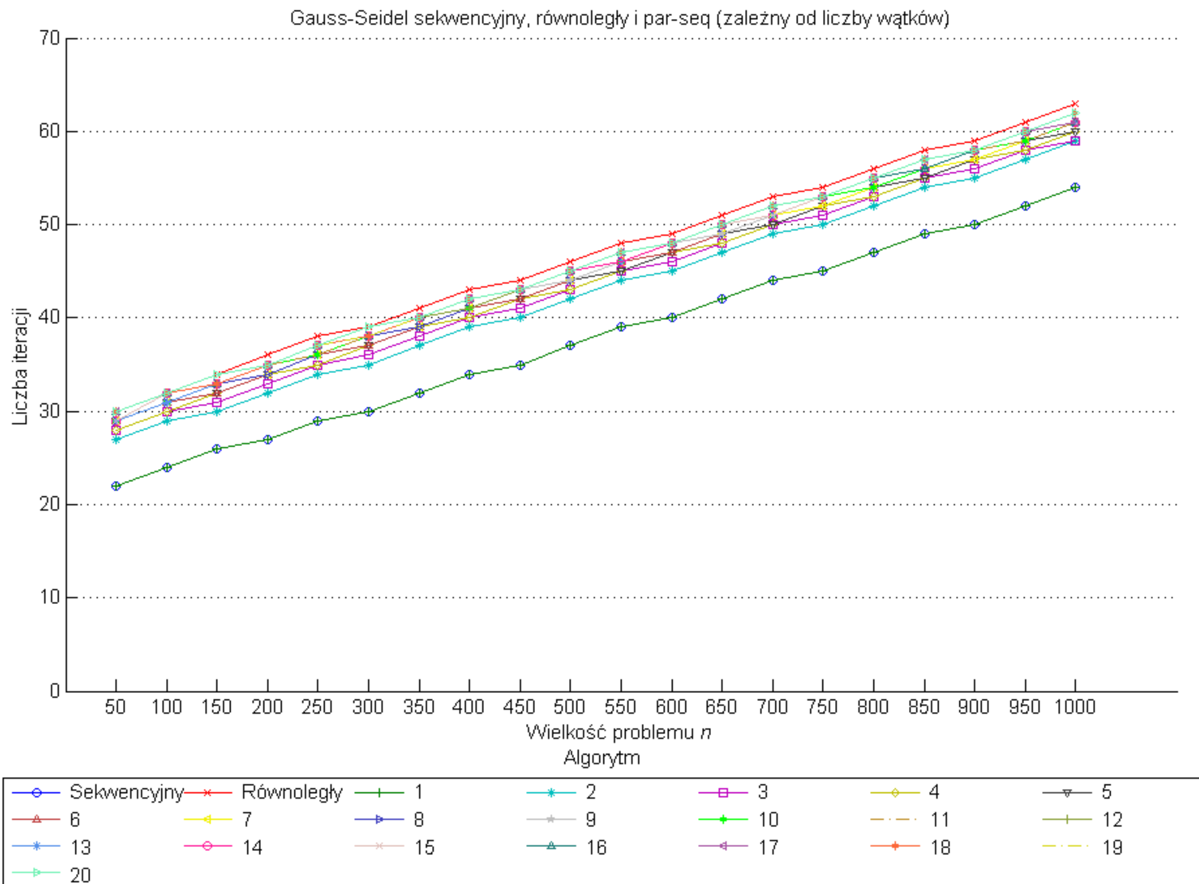
Rys. 3.1 Porównanie liczby iteracji sekwencyjnego i równoległego algorytmu Gaussa-Seidela dla funkcji 4 stopnia



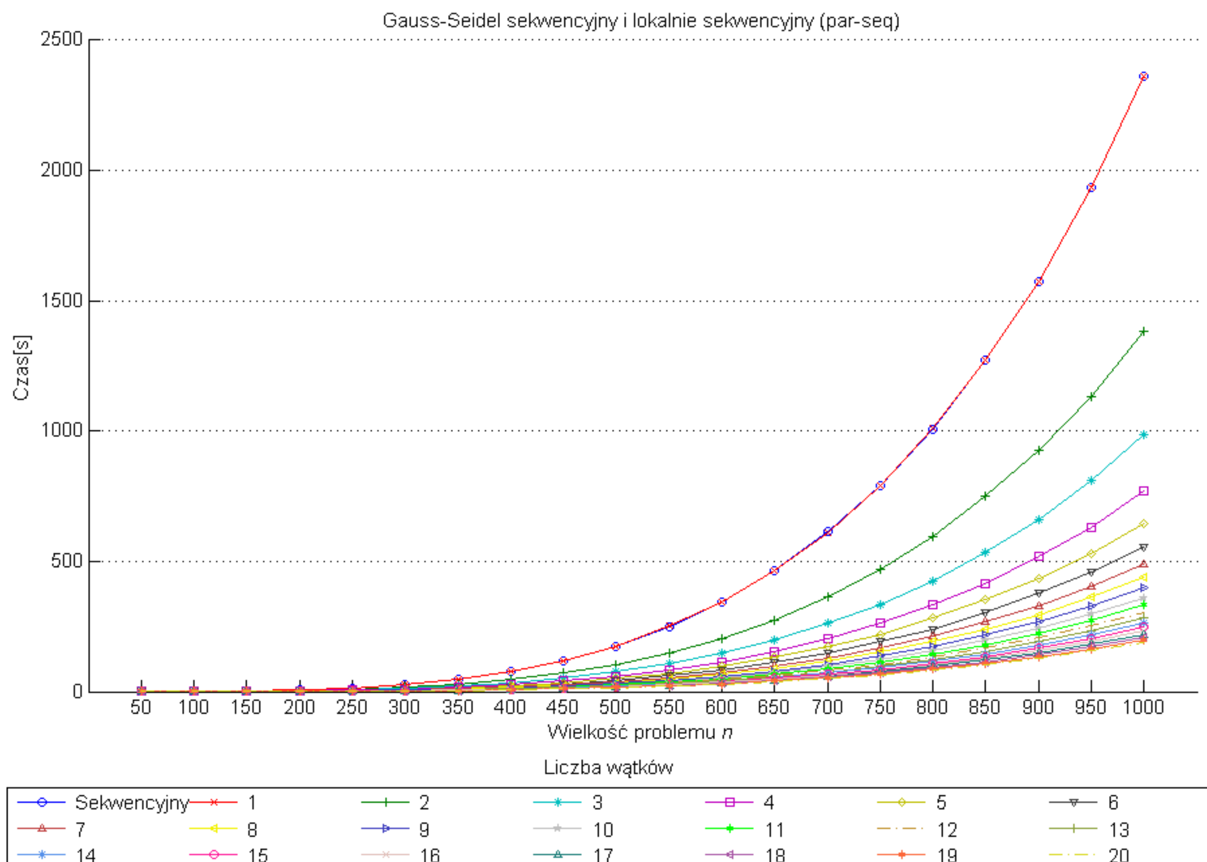
Rys. 3.2 Porównanie czasu obliczeń sekwencyjnego i równoległego algorytmu Gaussa-Seidela dla funkcji 4 stopnia



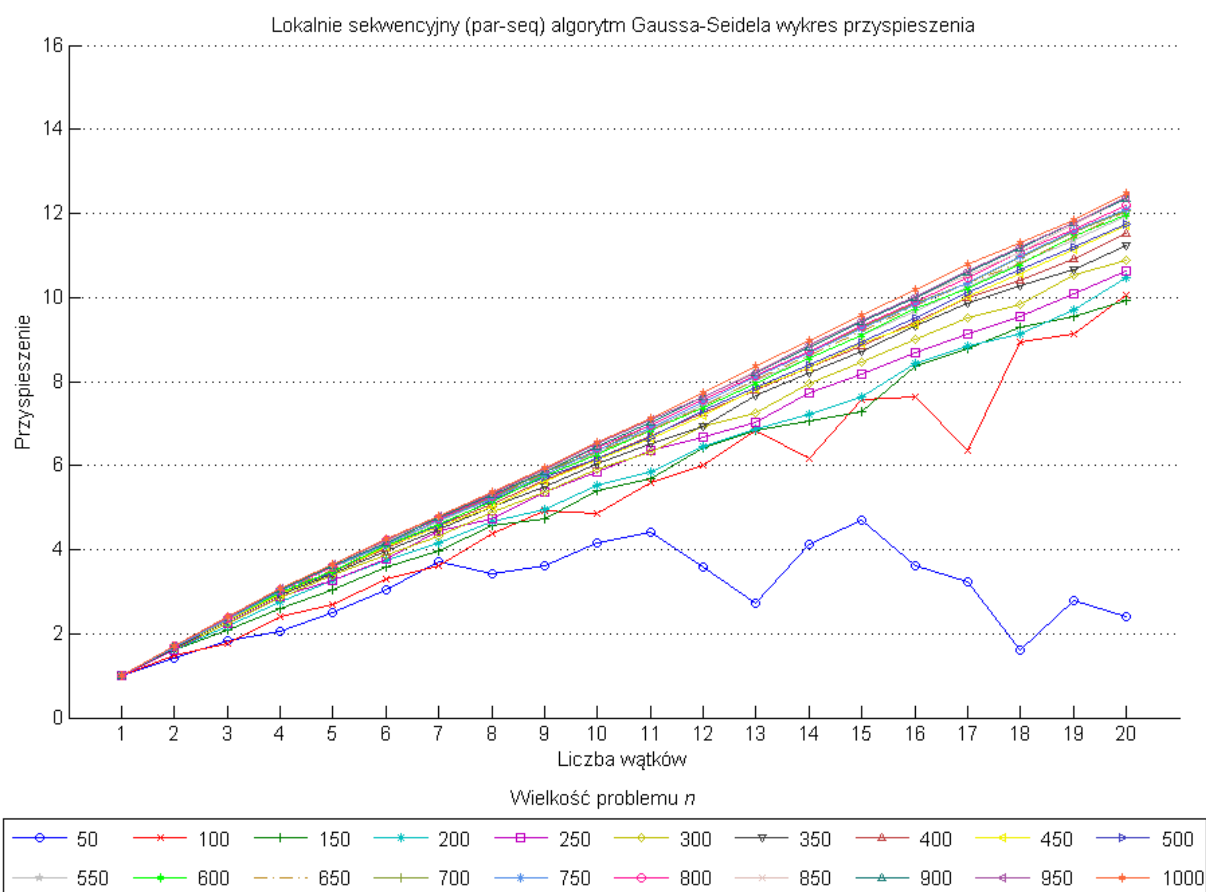
Rys. 3.3 Przyspieszenie zrównoleglonej wersji algorytmu Gaussa-Seidela dla funkcji 4 stopnia



Rys. 3.4 Porównanie liczby iteracji sekwencyjnego, równoległego oraz lokalnie sekwencyjnego algorytmu Gaussa-Seidela



Rys. 3.5 Porównanie czasów obliczeń sekwencyjnego i lokalnie sekwencyjnego (par-seq) algorytmu Gaussa-Seidela dla funkcji 4 stopnia



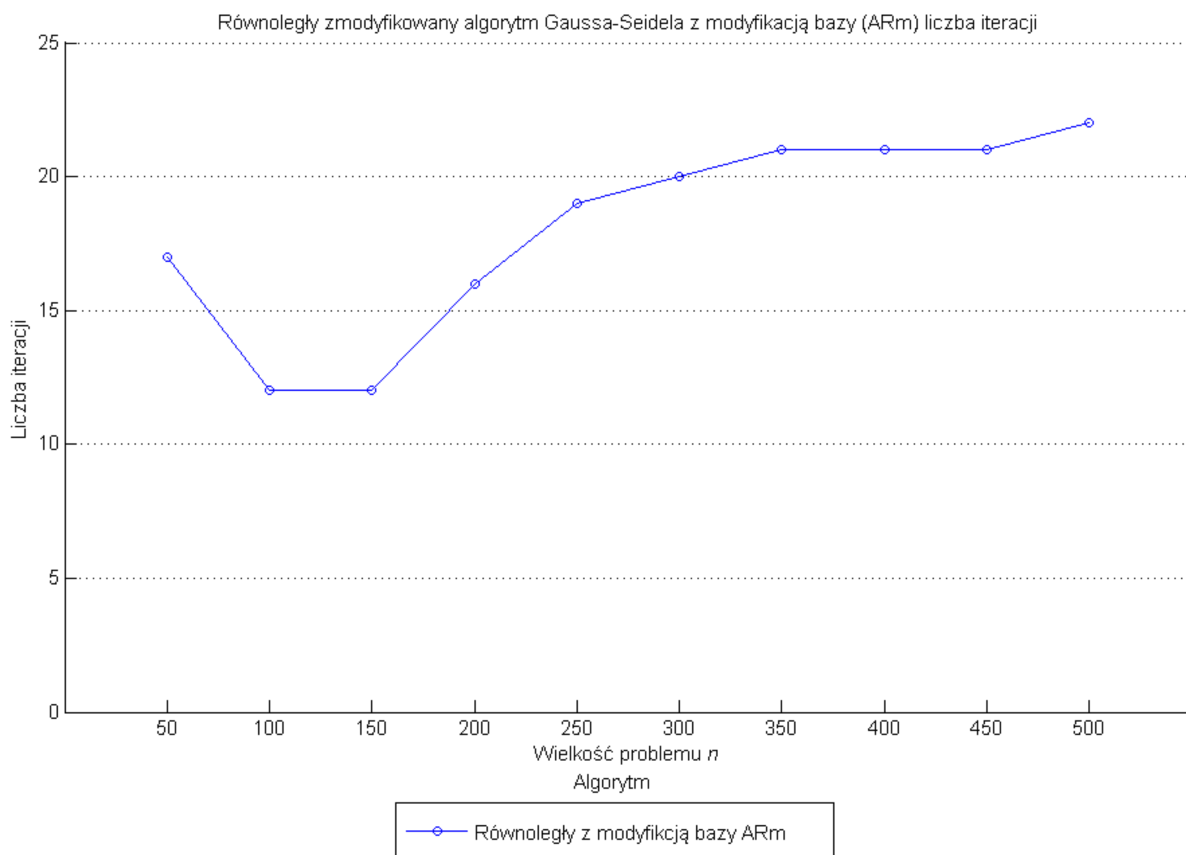
Rys. 3.6 Przyspieszenie lokalnie sekwencyjnej (par-seq) wersji algorytmu Gaussa-Seidela dla funkcji 4 stopnia

Otrzymane wyniki zaprezentowane na wykresach (od Rys. 3.1 do Rys. 3.6) pokazują, że algorytm sekwencyjny potrzebował najmniej iteracji do obliczenia minimum, natomiast równoległa wersja potrzebowała ich najwięcej. Algorytm w wersji par-seq dla 1 wątku działa tak samo jak algorytm sekwencyjny, dlatego oba algorytmy potrzebują taką samą liczbę iteracji (linia niebieska (sekwencyjny) znajduje się pod linią zieloną – 1 wątek). Wraz ze wzrostem liczby użytych wątków liczba iteracji wzrasta, aż do liczby iteracji potrzebnych do odnalezienia minimum przez algorytm równoległy. Algorytm par-seq uruchomiony na liczbie wątków równej wielkości problemu n będzie działał tak jak algorytm równoległy, to jest każda optymalizacja kierunkowa będzie obliczana niezależnie. W takim przypadku liczba iteracji dla obu algorytmów jest taka sama, jednak z obliczeń wynika, że zrównanie liczby iteracji następuje znacznie wcześniej.

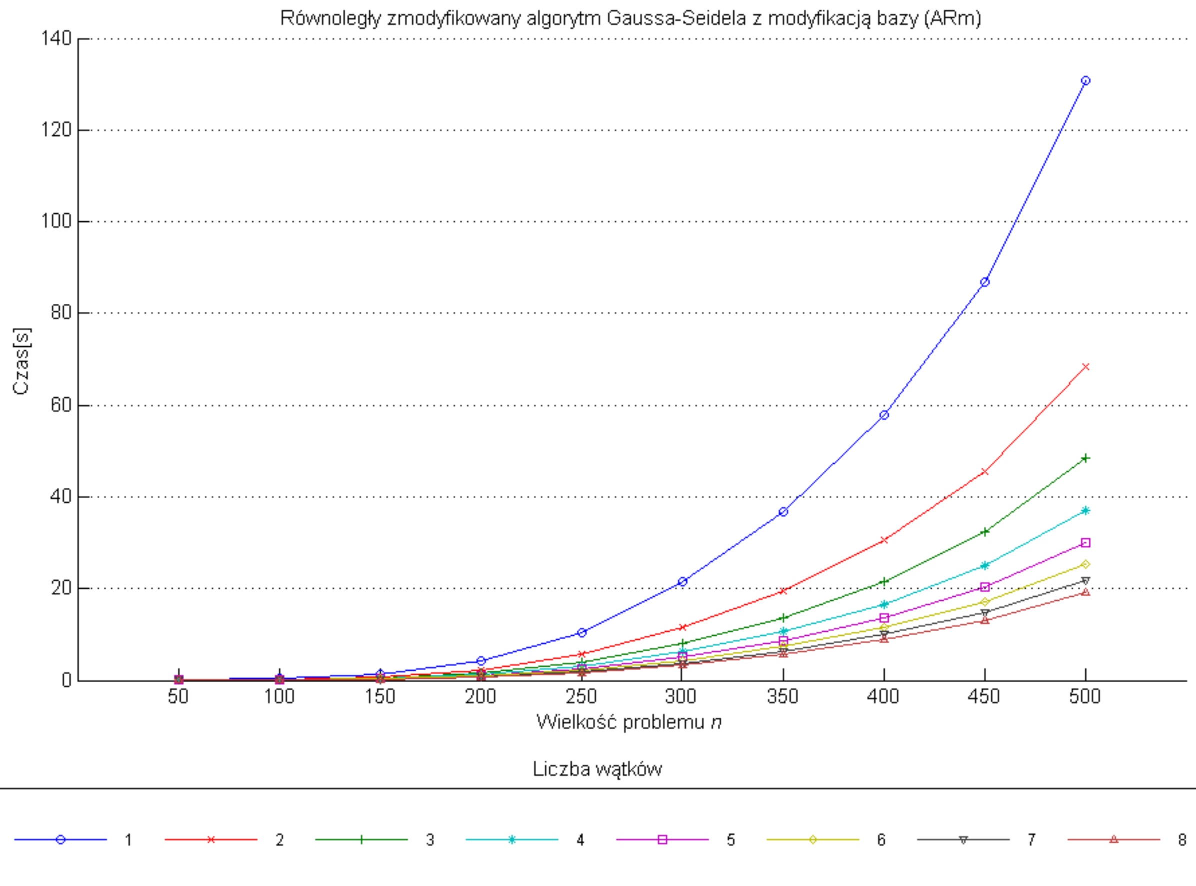
3.2 Z modyfikowany równoległy algorytm Gaussa-Seidela z modyfikacją bazy (ARm)

Obliczenia dla zmodyfikowanego równoległego algorytmu Gaussa-Seidela z modyfikacją bazy (ARm) zostały przeprowadzone dla tych samych funkcji testowych, które zostały wykorzystane do przeprowadzenia obliczeń dla sekwencyjnej i równoległej wersji algorytmu Gauss-Seidela. Obliczenia zostały wykonane na komputerze z 8 rdzeniowym procesorem AMD RYZEN 7 5800X dla wielkości problemu *n* od 50 do 500.

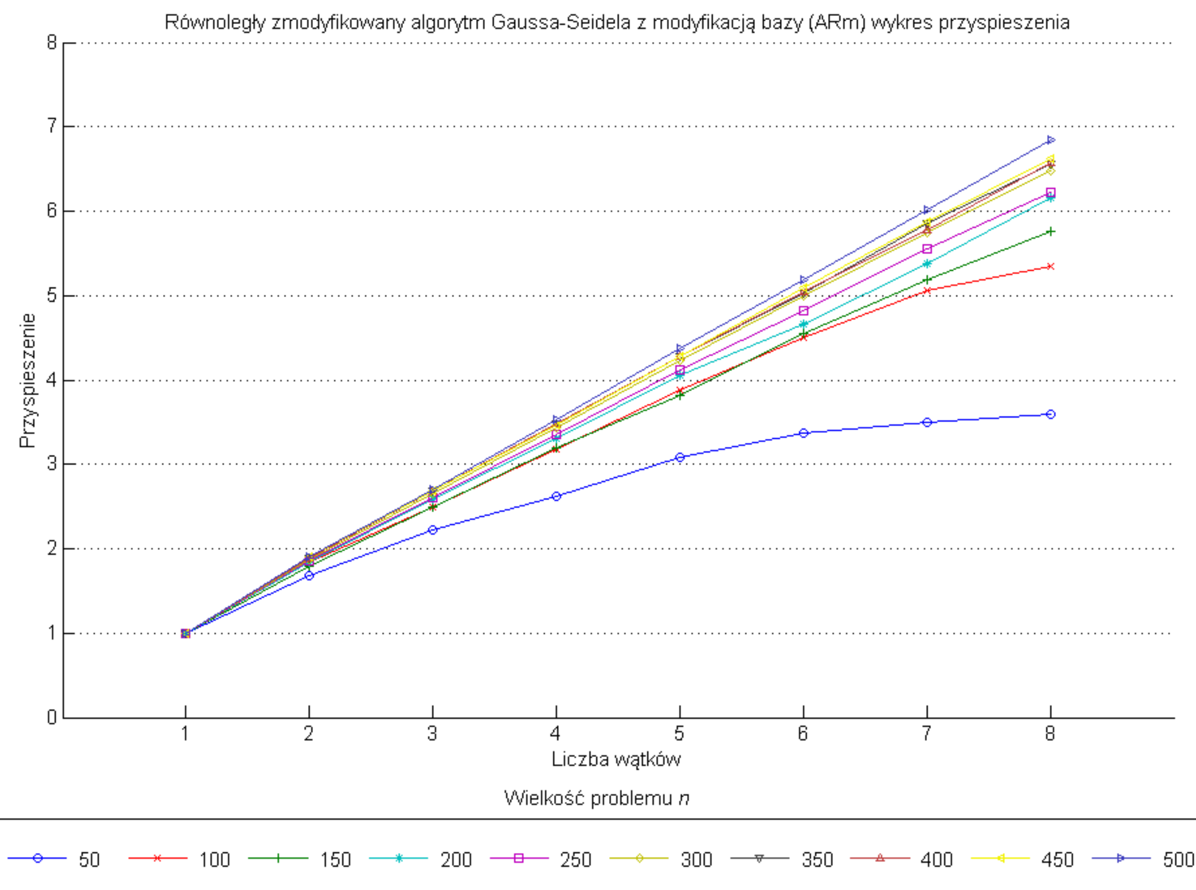
Natomiast dla funkcji 4 stopnia wyniki przedstawiono poniżej:



Rys. 3.7 Liczba iteracji równoległego algorytmu Gaussa-Seidela z modyfikacją bazy ARm



Rys. 3.8 Czas obliczeń równoległego algorytmu Gaussa-Seidela z modyfikacją bazy ARm



Rys. 3.9 Przyspieszenie równoległego algorytmu Gaussa-Seidela z modyfikacją bazy ARm

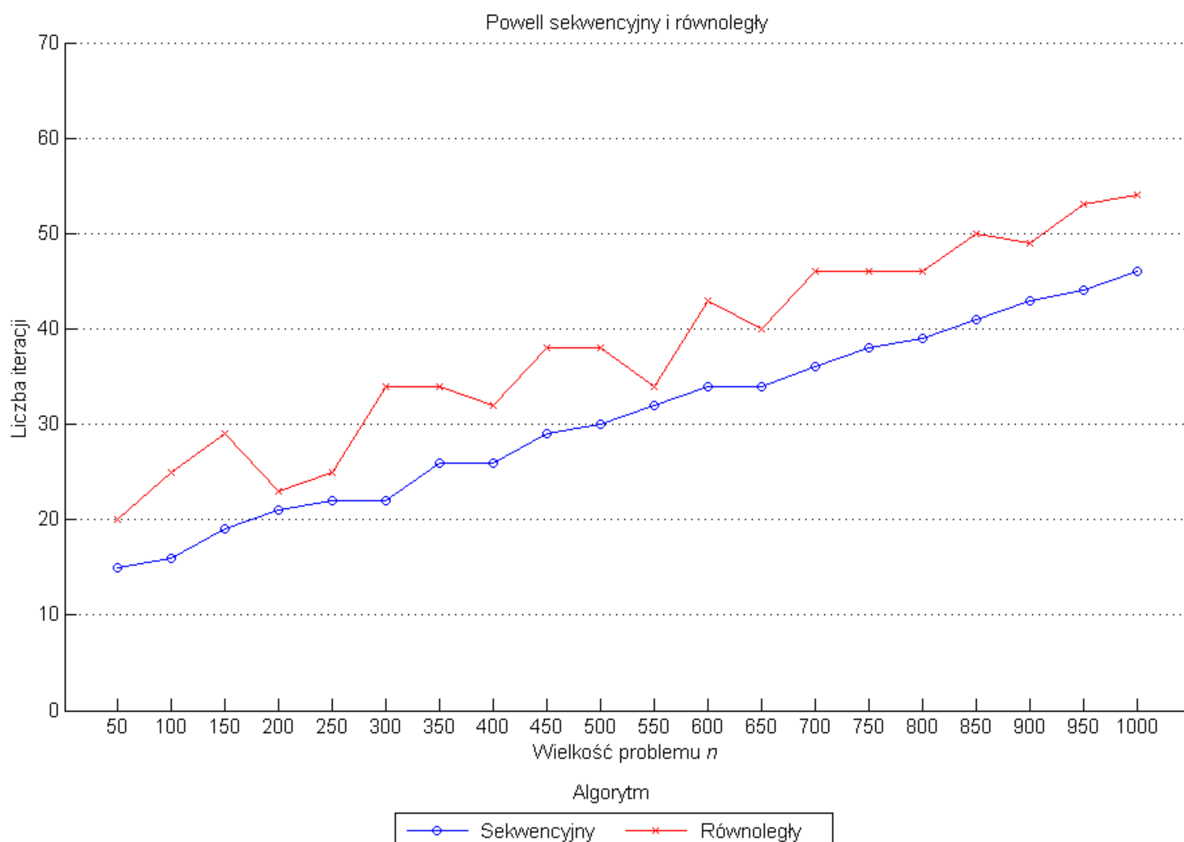
Wykres 3.8 przedstawia czasy obliczeń tej wersji algorytmu Gaussa-Seidela dla liczby wątków od 1 do 8, a wykres 3.9 prezentuje przyspieszenie tego algorytmu. Wykres 3.7 przedstawia liczbę iteracji potrzebą do obliczenia minimum, można zauważyć, że występują tutaj wahania. Związane jest to z wprowadzanymi modyfikacjami bazy kierunków oraz odświeżaniem tej bazy.

3.3 Algorytm Powella

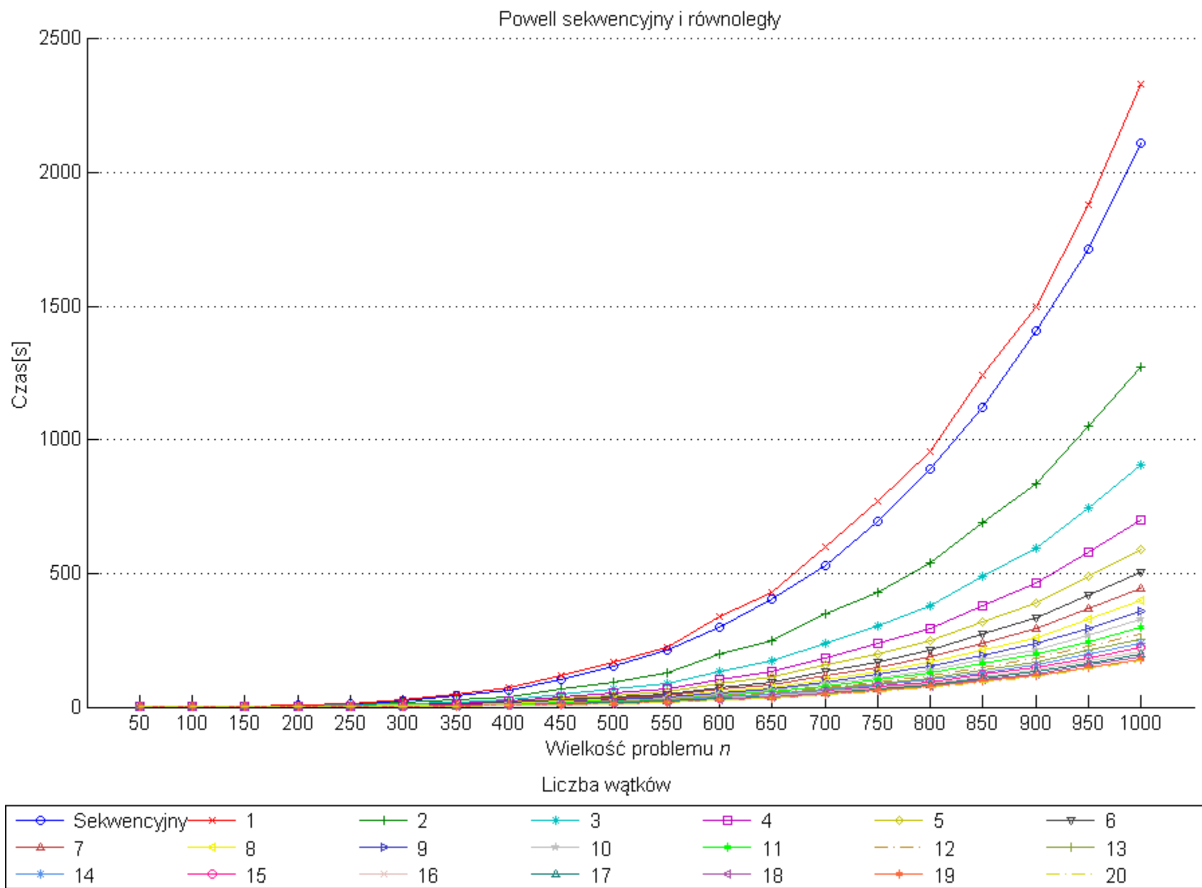
Drugim testowanym algorytmem była metoda Powella. Przetestowano różne odmiany tego algorytmu między innymi algorytm bez modyfikacji bazy, algorytm z modyfikacją bazy oraz algorytm z przeszukiwaniem wiązką.

3.3.1 Algorytm Powella bez modyfikacji bazy

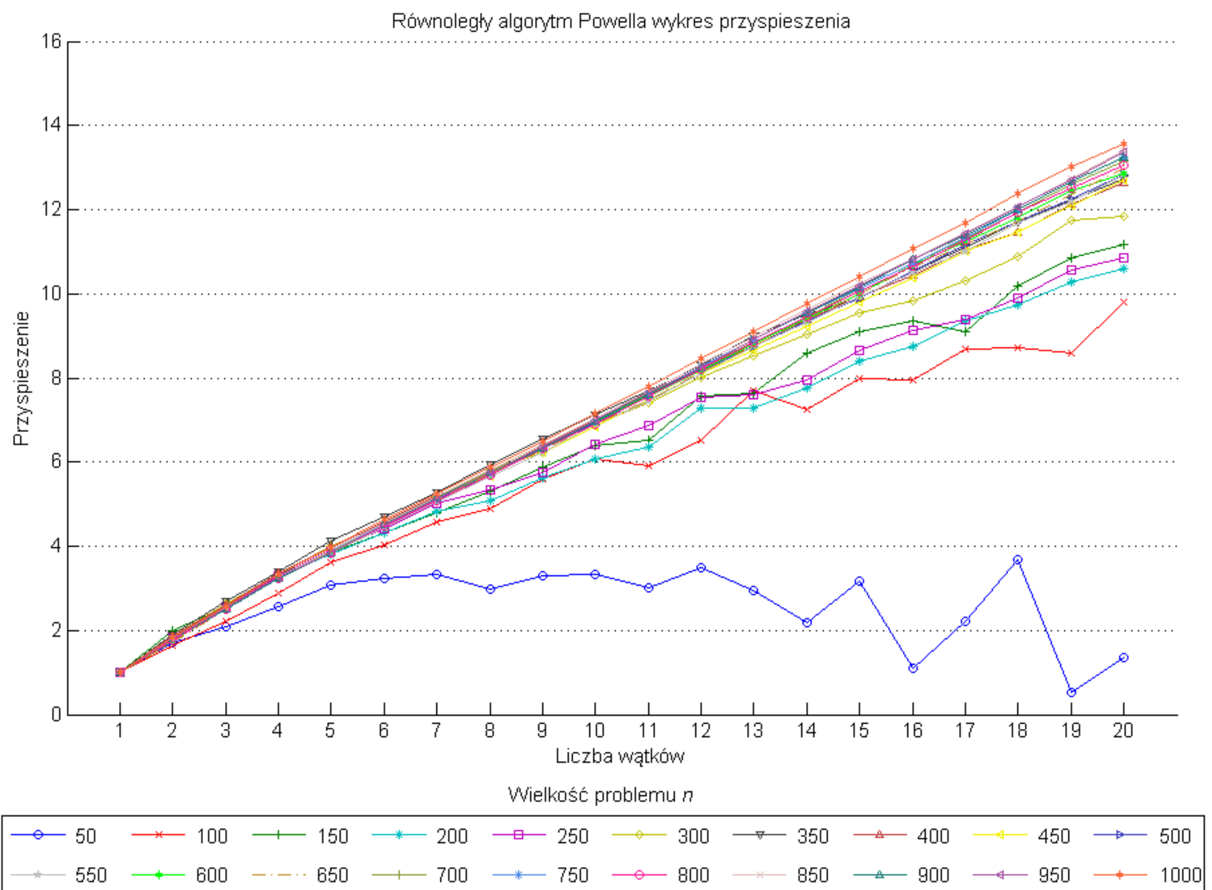
Algorytm Powella bez modyfikacji bazy (to znaczy nowy kierunek nie jest wprowadzany do bazy kierunków) został przetestowany dla wielkości problemu n od 50 do 1000 z krokiem 50. Wyniki dla drugiej funkcji testowej zaprezentowano poniżej:



Rys. 3.10 Porównanie liczby iteracji sekwencyjnego i równoległego algorytmu Powella dla funkcji 4 stopnia



Rys. 3.11 Porównanie czasu obliczeń sekwencyjnego i równoległego algorytmu Powella dla funkcji 4 stopnia



Rys. 3.12 Przyspieszenie zrównoleglonej wersji algorytmu Powella dla funkcji 4 stopnia

Otrzymane wyniki dla funkcji 4 stopnia (3.2) zaprezentowane na wykresach (do Rys. 3.10 do Rys. 3.12) przedstawiają podobną, jak w algorytmie Gauss-Seidela, zależność pomiędzy sekwencyjną i równoległa wersją algorytmu Powella to znaczy równoległa wersja potrzebuje większej liczby iteracji niż wersja sekwencyjna, co przekłada się na dłuższy czas obliczeń zrównoleglonego algorytmu wykonywanego na jednym wątku niż algorytmu sekwencyjnego. Jednakże równoległy algorytm może zostać uruchomiony na większej liczbie wątków, dzięki czemu czas obliczeń zostaje znacząco skrócony. Dla wielkości problemu *n* równego 50 występowały zachowania niepożądane, tzn. spadek przyśpieszenia.\

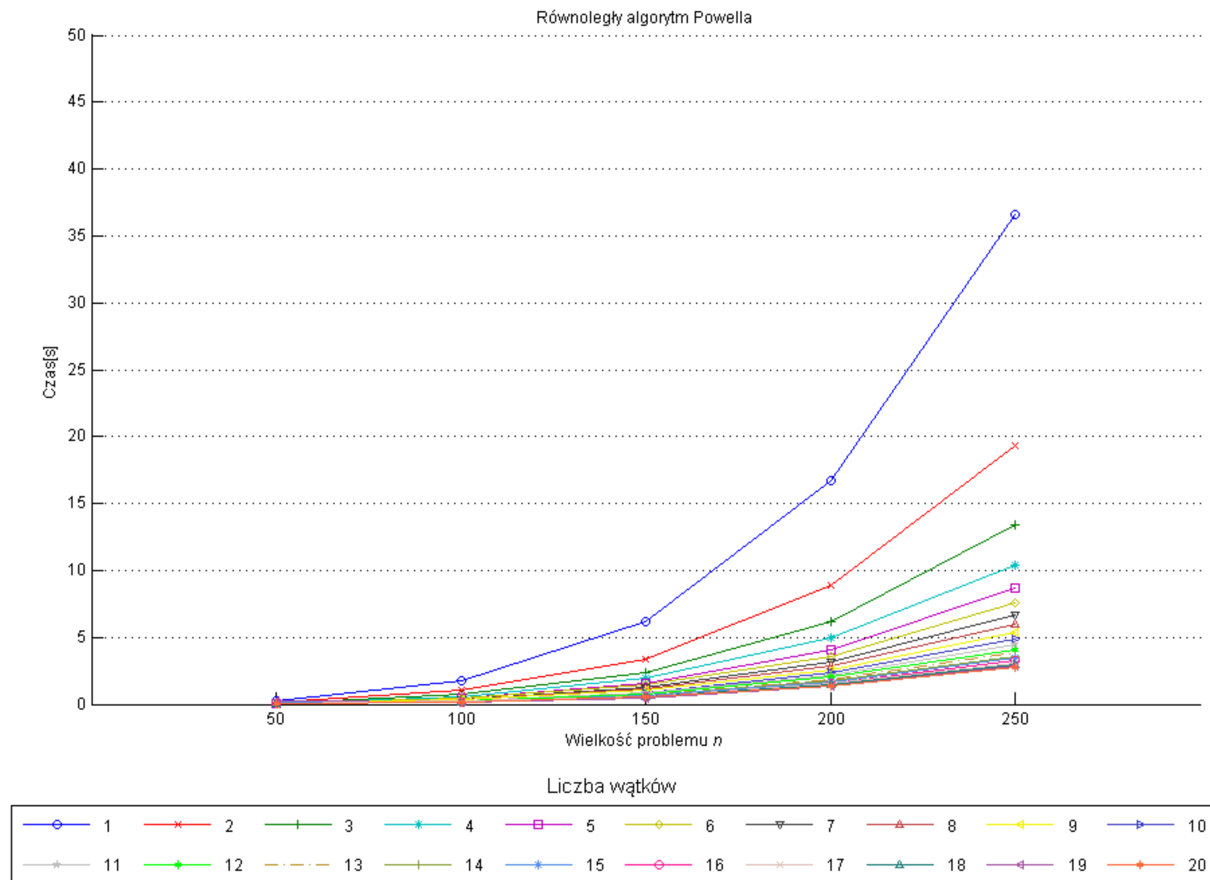
3.3.2 Algorytm Powella z modyfikacją bazy

Drugą wersją algorytmu Powella, dla której przeprowadzono testy, była wersja z modyfikacją bazy to znaczy nowy kierunek generowany w każdej iteracji był wprowadzany do bazy już istniejących kierunków po spełnieniu odpowiedniego warunku. Wprowadzanie kierunku do bazy kierunków prowadziło do jej nieortogonalności. Dla tej wersji algorytmu przeprowadzono badania dla dwóch pierwszych funkcji testowych w zakresie wielkości problemu *n* od 50 do 250 z krokiem 50. Badania nie zostały przeprowadzone dla większych wielkości problemu, ponieważ w żadnym z tych przypadków nie został spełniony warunek zezwalający na wprowadzenie kierunku do bazy. Skutkowało to tym, że w badanych przypadkach ta wersja algorytmu zachowywała się tak samo jak algorytm bez modyfikacji bazy. Do modyfikacja bazy dochodziło jedynie w przypadku trzeciej funkcji testowej to jest funkcji Rosenbrocka (3.3), dlatego nie przedstawiono wykresów dla testowej funkcji (3.2).

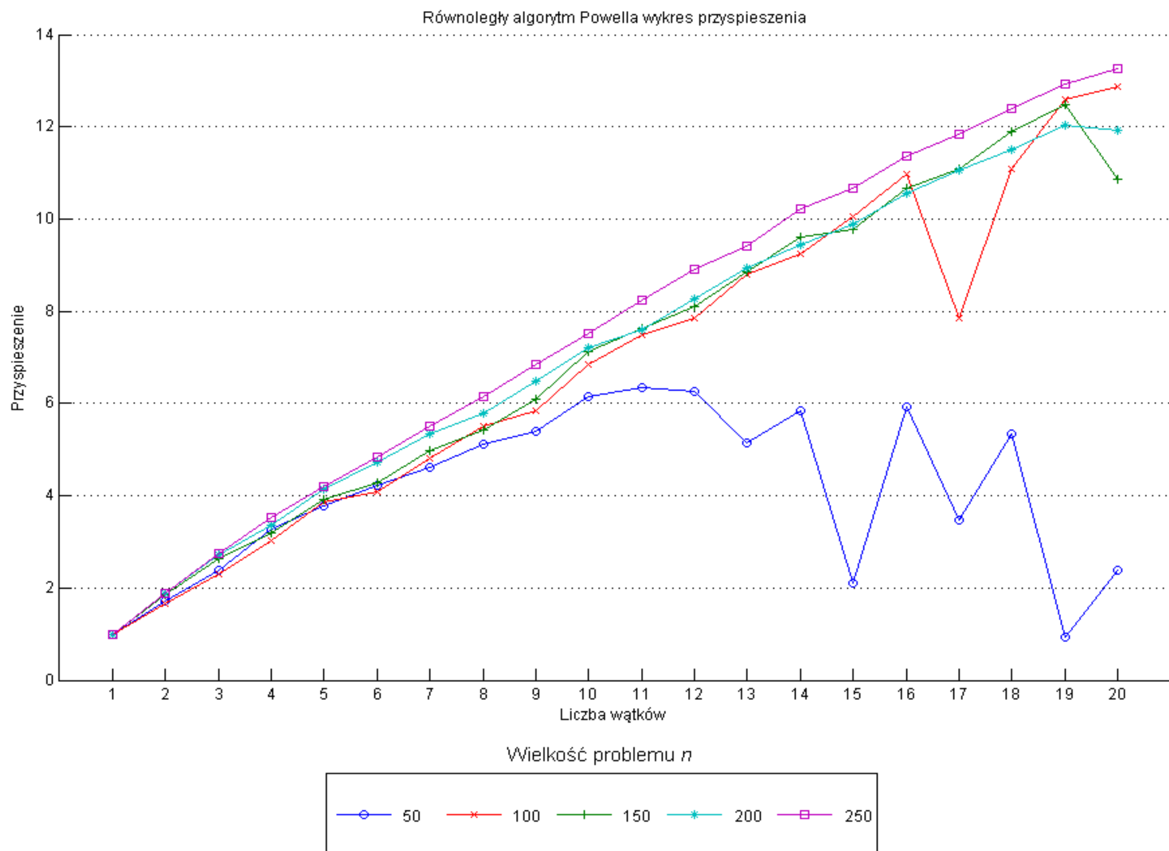
3.3.3 Algorytm Powella z przeszukiwaniem wiązką

Kolejną wersją algorytmu Powella, która została przetestowana była wersja z przeszukiwaniem wiązką na końcu każdej iteracji. Nowy kierunek nie był wprowadzany do bazy, służył jedynie do prowadzenia minimalizacji wzdłuż niego. Dla tej wersji algorytmu przeprowadzono badania w zakresie wielkości problemu *n* od 50 do 250 z krokiem 50.

Wyniki dla drugiej funkcji testowej (3.2):



Rys. 3.13 Porównanie czasu obliczeń równoległego algorytmu Powella dla funkcji 4 stopnia



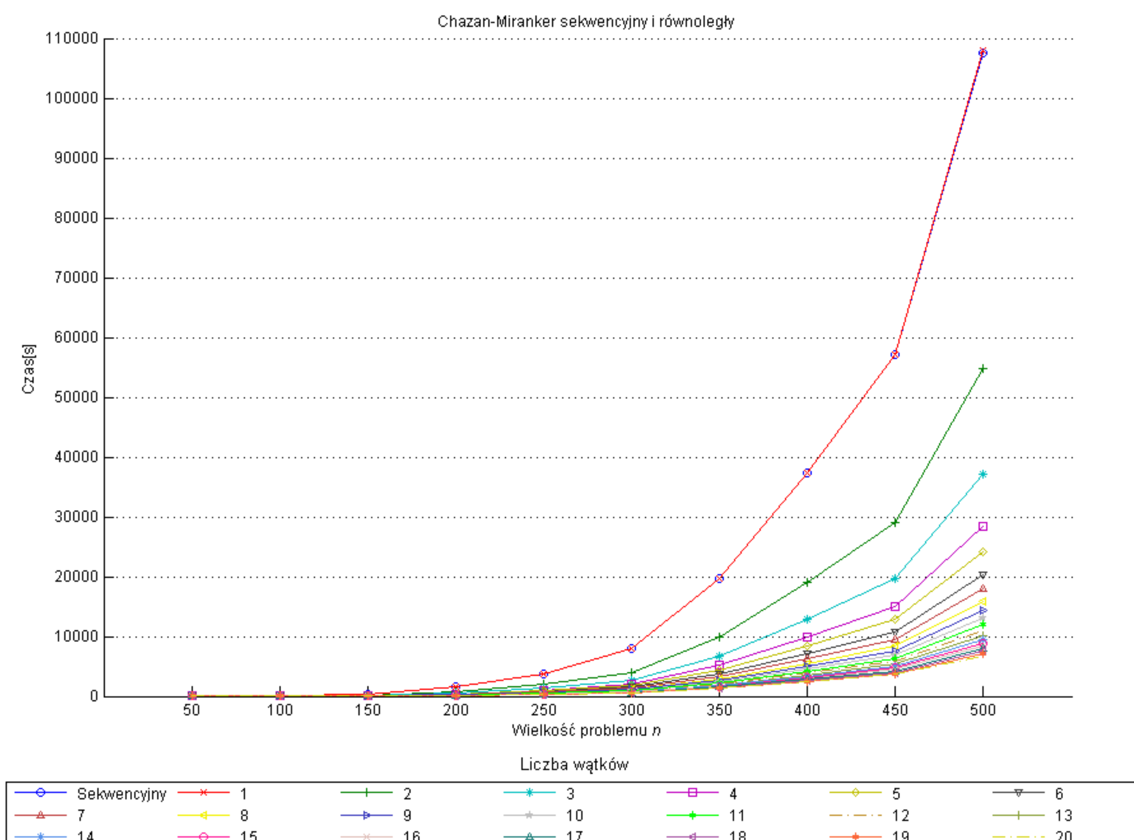
Rys. 3.14 Przyspieszenie zrównoleglonej wersji algorytmu Powella dla funkcji 4 stopnia

Wyniki zaprezentowane na wykresach (Rys. 3.13 i Rys. 3.14) pokazują, że algorytm Powella z przeszukiwaniem wiązką dla danego przedziału wielkości problemu osiąga dobre wyniki przyśpieszenia. Na Rys. 3.14 dla wymiaru problemu $n = 50$ przyśpieszenie wykazywało pewne wahania.

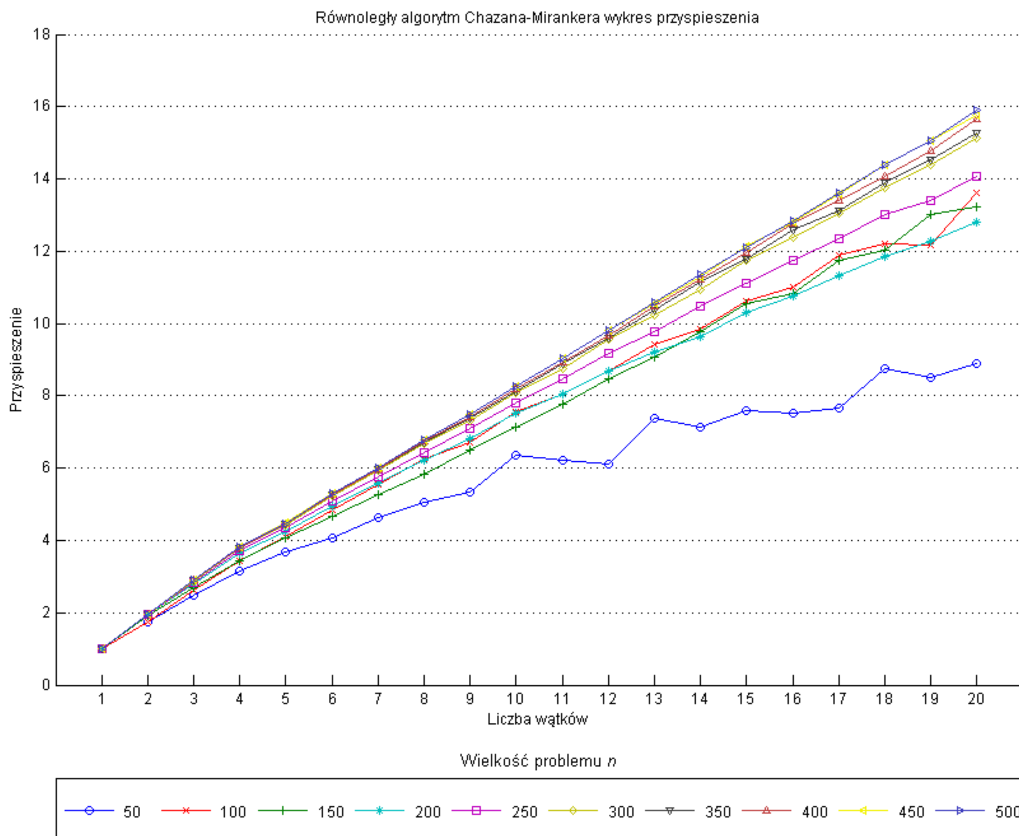
3.4 Algorytm Chazana-Mirankera

Następnym algorytmem, dla którego przeprowadzono badania, był algorytm Chazana-Mirankera. Algorytm ten posiada strukturę dopasowaną do obliczeń przeprowadzanych równolegle, ponieważ wersja sekwencyjna i wersja równoległa wykonywana na 1 wątku działają tak samo nie zamieszczono wykresów porównujących obie wersje (na wykresach dotyczących czasu wykonywania algorytmu zostały zamieszczone obie wersje). Ta metoda potrzebuje dużo iteracji do znalezienia minimum, co przekłada się na dość wolny czas wykonywania algorytmu (w porównaniu do innych algorytmów w tej pracy). Z tego powodu obliczenia przeprowadzono w zakresie wielkości problemu n od 50 do 500.

Dla funkcji czwartego stopnia (3.2) otrzymano następujące wyniki:



Rys. 3.15 Porównanie czasu obliczeń sekwencyjnego i równoległego algorytmu Chazana-Mirankera dla funkcji 4 stopnia



Rys. 3.16 Przyspieszenie zrównoleglonej wersji algorytmu Chazana-Mirankera dla funkcji 4 stopnia

Wykresy (Rys. 3.15 i Rys. 3.16) przedstawiają porównanie czasów obliczeń zależnie od liczby użytych wątków oraz przyśpieszenia uzyskane dla nich. Wykres przyśpieszenia dla wielkości problemu *n* równego 50 podobnie, jak to było we wcześniejszych algorytmach, wykazuje pewną „niestabilność". Spowodowane jest to jak poprzednio nierównomiernym podziałem zadań. Zgodnie z założeniem budowa algorytmu dopasowana do równoległej pracy osiągała ten sam czas dla wersji sekwencyjnej oraz wersja równoległej wykonywanej na 1 wątku.

3.5 Algorytm Neldera-Meada

Algorytm Neldera-Meada [25, 27, 29] jest metodą wyznaczania ekstremum (minimum/maksimum) ogólnie nieliniowych funkcji $f(\boldsymbol{x})$ określonych w przestrzeni R^n. Głównym jego założeniem jest wygenerowanie i przekształcanie sympleksu, który jest *n* wymiarowy i posiada *n*+1 wierzchołków. Dla wymiaru problemu *n* równego 2 przyjęty w pracy początkowy sympleks zbudowany z 3 wierzchołków miał współrzędne (0,0), (0,1) oraz (1,1). Sympleks poddawany jest 4 rodzajom operacji, te operacje to odbicie (3.4), ekspansja (3.5),

redukcja (3.6), a także zawężanie (3.7) (inaczej zwane kontrakcją). W klasycznej sekwencyjnej wersji algorytmu operacje na sympleksie dokonywane są względem środka ciężkości dla węzła, w którym wartość funkcji jest największa, przy założeniu, że poszukiwane jest minimum funkcji. Wykonywanie operacji na sympleksie trwa dopóki nie zostanie spełnione kryterium stopu. Kryterium stopu dla tego algorytmu opiera się na sukcesywnym wyliczaniu maksymalnej odległości pomiędzy wierzchołkami sympleksu (lub minimalnego promienia *n*-wymiarowej kuli zawierającej wszystkie wierzchołki sympleksu), dopóki jej wartość nie będzie mniejsza niż przyjęta dokładność - wtedy algorytm ulega zakończeniu.

W celu sformułowania algorytmu Neldera-Meada zastosowano poniższe oznaczenia:

n - jest wymiarem problemu, dla którego przeprowadzana jest optymalizacja,

$\boldsymbol{p}^{i}$ - n+1wierzchołków (gdzie i=1,2,...,n,n+1) dla n-wymiarowego sympleksu

$\boldsymbol{p}^{max}$ - oznaczenie wierzchołka, dla którego minimalizowana funkcja osiąga wartość największą,

$\overline{\boldsymbol{p}}$ - środek symetrii sympleksu z pominięciem wierzchołka $\boldsymbol{p}^{max}$.

Biorąc pod uwagę te oznaczenia operacje przekształcenia sympleksu dla tradycyjnej sekwencyjnej wersji algorytmu można zapisać w poniższy sposób:

1. operacja odbicia:

$$\boldsymbol{p}^{odb} = \overline{\boldsymbol{p}} + \alpha(\overline{\boldsymbol{p}} - \boldsymbol{p}^{max}) \quad (3.4)$$

2. operacja ekspansji:

$$\boldsymbol{p}^{eks} = \overline{\boldsymbol{p}} + \gamma(\boldsymbol{p}^{odb} - \overline{\boldsymbol{p}}) \quad (3.5)$$

3. operacja kontrakcji:

$$\boldsymbol{p}^{z} = \overline{\boldsymbol{p}} + \beta(\boldsymbol{p}^{max} - \overline{\boldsymbol{p}}) \quad (3.6)$$

4. operacja redukcji:

$$\boldsymbol{p}^{i} = \delta(\boldsymbol{p}^{i} + \boldsymbol{p}^{min}), \quad (3.7)$$

gdzie: $i = 0, 1, 2, \ldots, n$

$i \neq \min$

Współczynniki wykorzystywane w tych przekształceniach miały, w początkowej fazie, wartości:

α (alfa) – odbicia = 1,00

β (beta) –kontrakcji = 0,50

γ (gamma) – ekspansji = 2,00

δ (delta) – redukcji = 0,50

Badania wykazały, że niektórych przypadkach wartości te nie dają poprawnego rozwiązania. Ten problem występował głównie dla większych wielkości problemu dla algorytmu sekwencyjnego, dlatego eksperymentalnie wyznaczono 3 zestawy współczynników:

- $\alpha = 1{,}00$, $\beta = 0{,}80$, $\gamma = 2{,}00$, $\delta = 0{,}50$
- $\alpha = 1{,}00$, $\beta = 0{,}80$, $\gamma = 1{,}20$, $\delta = 0{,}50$
- $\alpha = 1{,}00$, $\beta = 0{,}80$, $\gamma = 1{,}10$, $\delta = 0{,}50$.

Algorytm Neldera-Meada w równoległej wersji został sformułowany przez Virginię Torczon [7, 38]. W tej wersji przekształcenia sympleksu dokonywane są na węzłach sympleksu podobnie jak w wersji sekwencyjnej, ale z pewnymi różnicami. Nie występuje tutaj operacja redukcji. Kolejność wykonywanych operacji jest analogiczna jak w wersji sekwencyjnej, oprócz redukcji, która nie występuje. Przekształcenia wykonywane są na n węzłach sympleksu naraz. Jednakże w odróżnieniu od wersji sekwencyjnej, gdzie operacje wykonywane były względem środka symetrii, przekształcanie następuje względem węzła, w którym minimalizowana funkcja osiąga wartość najmniejszą. W związku z tym w inny sposób następuje sprawdzenie pomyślności operacji. W celu sprawdzenia czy dana operacja jest wykonana pomyślnie wylicza się wartości funkcji we wszystkich węzłach sympleksu, a następnie porównuje się najmniejszą spośród tych wartości z najmniejszą wartością funkcji przed tym przekształceniem, jeżeli ta wartość jest mniejsza to operacja uznawana jest za poprawną. Opisane cechy algorytmu w takiej wersji równoległej będą powodować generowanie innych sympleksów w trakcie przeprowadzania obliczeń niż jak to jest w algorytmie sekwencyjnym.

Do zapisania powyższej wersji algorytmu oraz operacji przeprowadzanych na sympleksie przyjęto poniższe oznaczenia:

$\boldsymbol{p}^{i}$ - n+1wierzchołków (gdzie $i = 0, 1, 2, ..., n$) dla n-wymiarowego sympleksu

$\boldsymbol{p}^{min}$ - oznaczenie wierzchołka, w którym minimalizowana funkcja osiąga wartość najmniejszą.

Na podstawie powyższych przyjętych oznaczeń operacje wykonywane na sympleksie można zapisać w następujący sposób:

1. operacja odbicia:

$$\boldsymbol{p}_i^{\text{odb}} = \boldsymbol{p}^{\min} + \alpha\left(\boldsymbol{p}^{\min} - \boldsymbol{p}^i\right), \tag{3.8}$$

gdzie: $i = 0, 1, 2, \ldots, n$

$i \neq \min$

2. operacja ekspansji:

$$\boldsymbol{p}_i^{\text{eks}} = \boldsymbol{p}^{\min} + \gamma\left(\mathbf{p}^{\min} - \boldsymbol{p}_i^{\text{odb}}\right), \tag{3.9}$$

gdzie: $i = 0, 1, 2, \ldots, n$

$i \neq \min$

3. operacja kontrakcji:

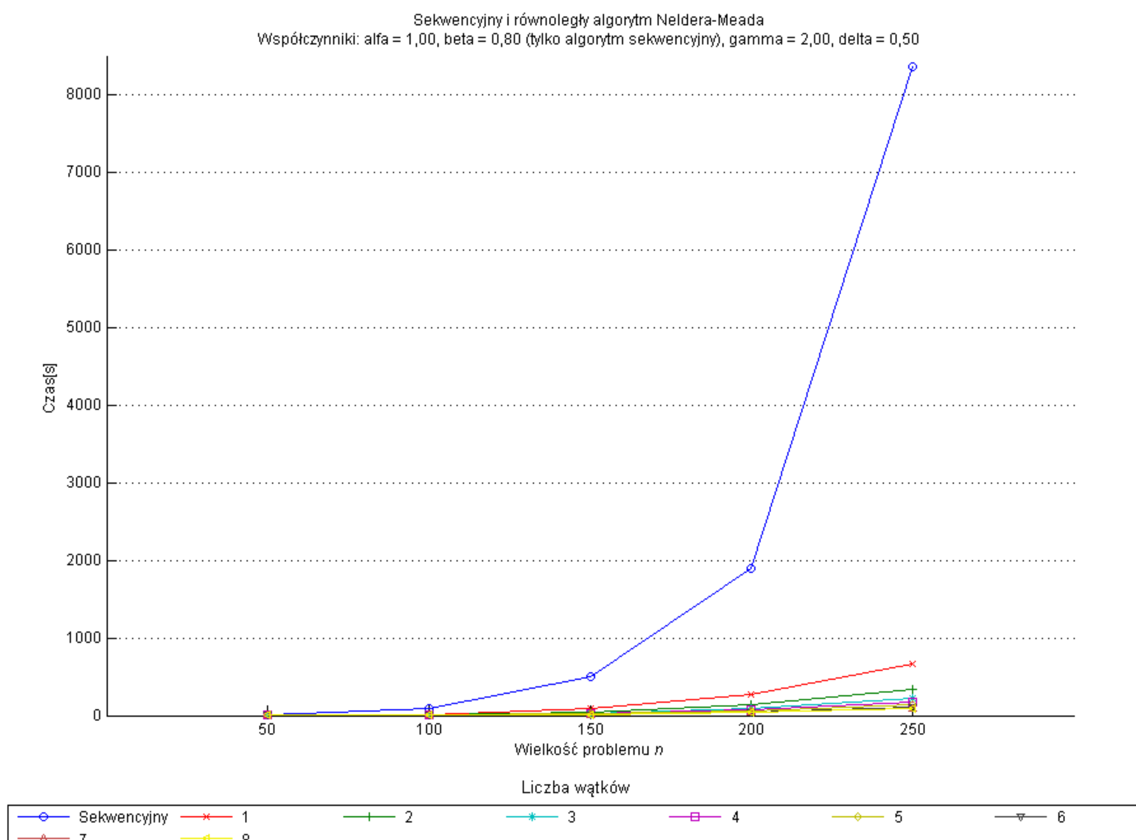
$$\boldsymbol{p}_i^{\text{z}} = \boldsymbol{p}^{\min} + \beta\left(\boldsymbol{p}^i - \boldsymbol{p}^{\min}\right), \tag{3.10}$$

gdzie: $i = 0, 1, 2, \ldots, n$

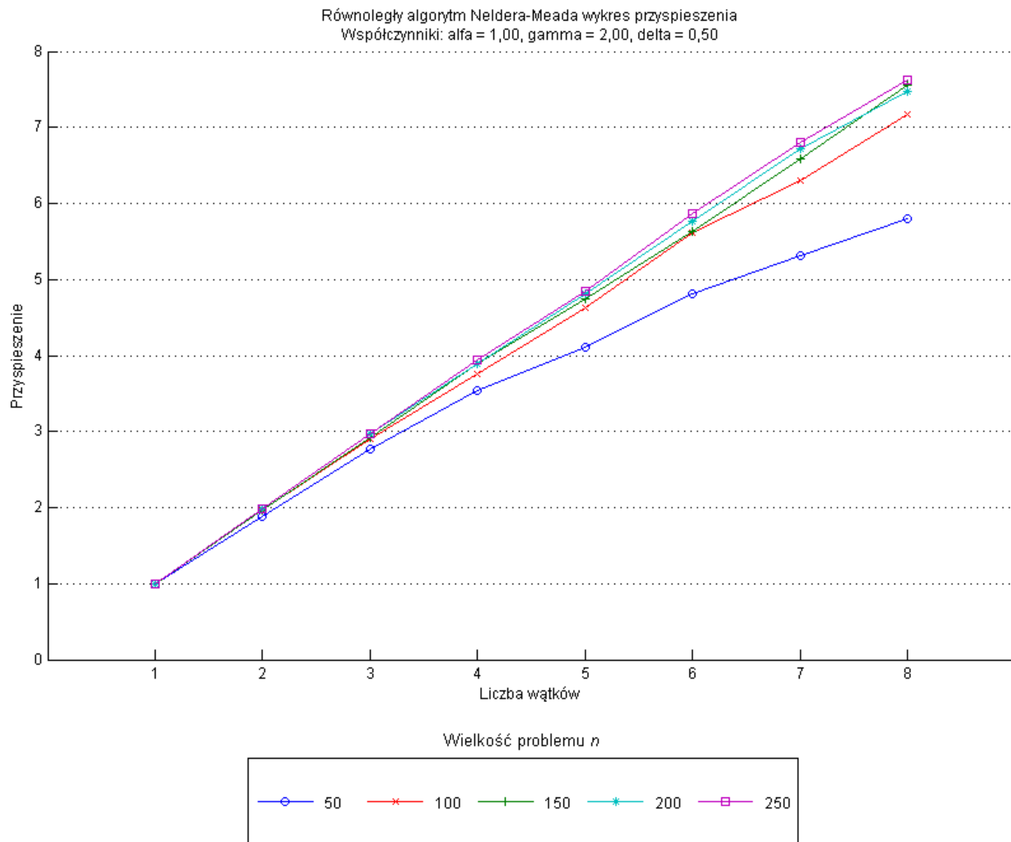
$i \neq \min$

Obliczenia przeprowadzono na komputerze 8 rdzeniowym.

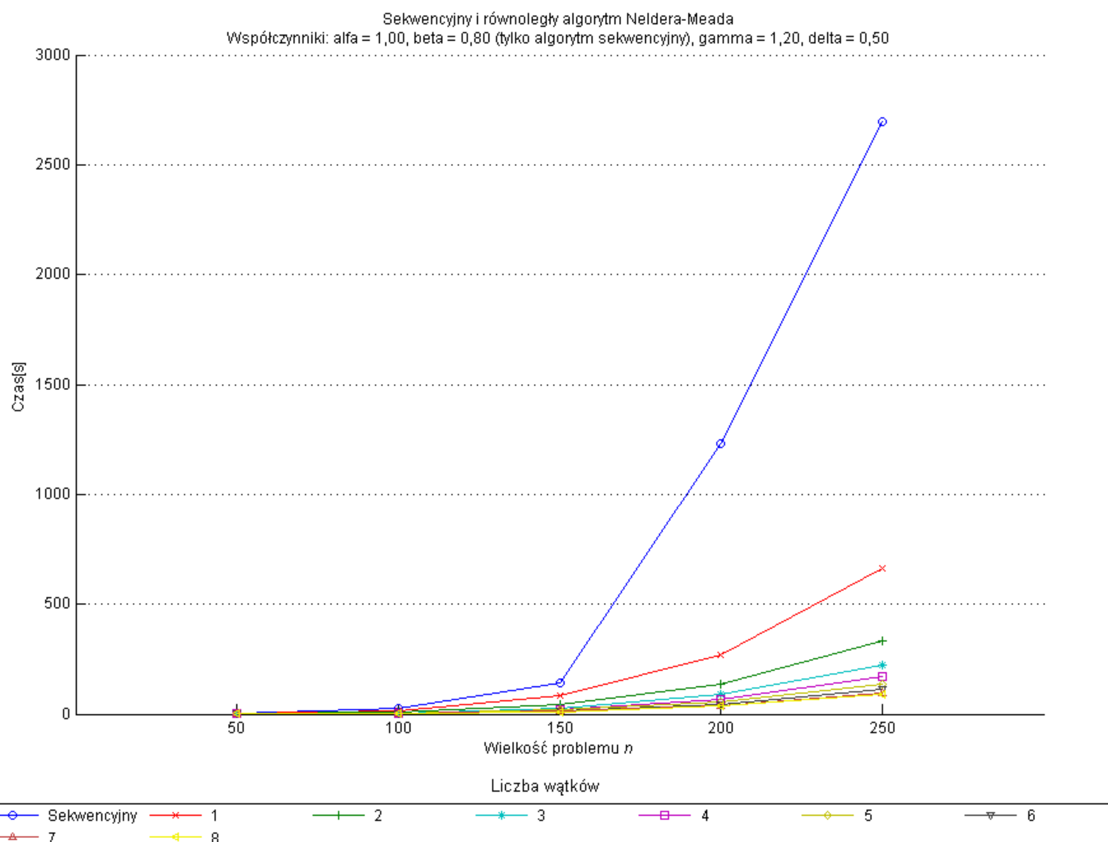
Dla drugiej funkcji testowej otrzymano następujące wyniki:



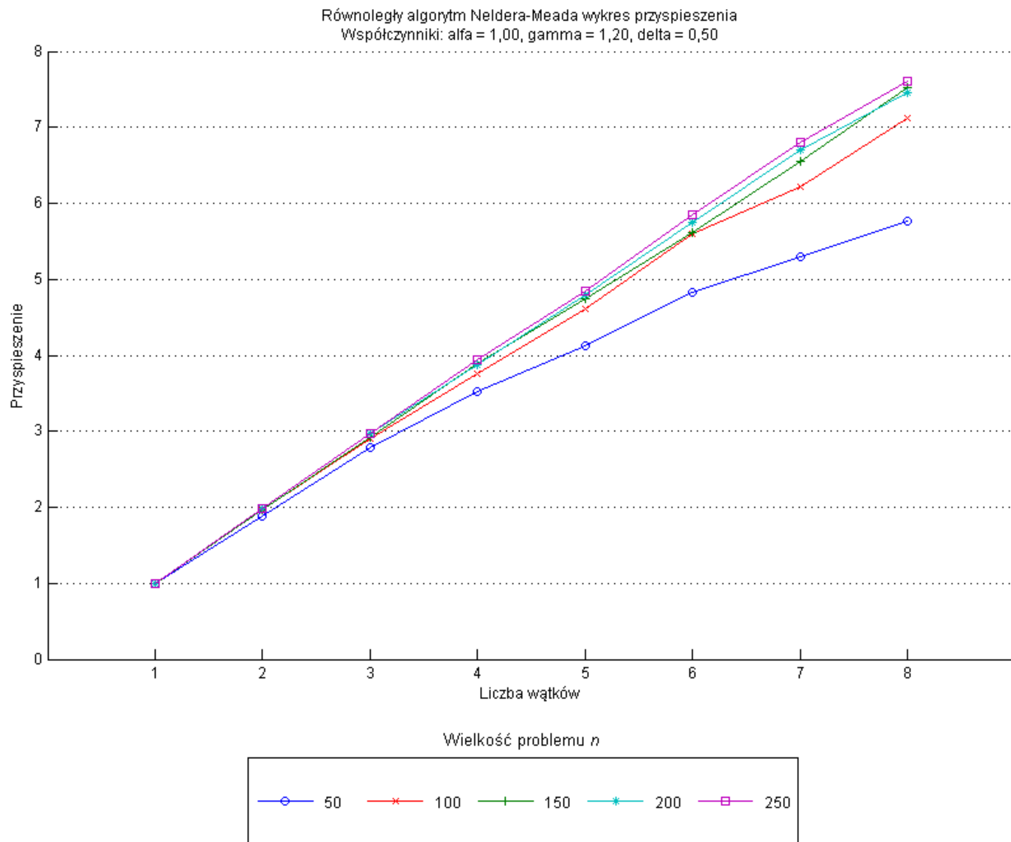
Rys. 3.17 Porównanie czasu obliczeń sekwencyjnego i równoległego algorytmu Neldera-Meada dla 2 funkcji testowej. Dla współczynników alfa= 1,00, beta= 0,80, gamma= 2,00, delta= 0,50



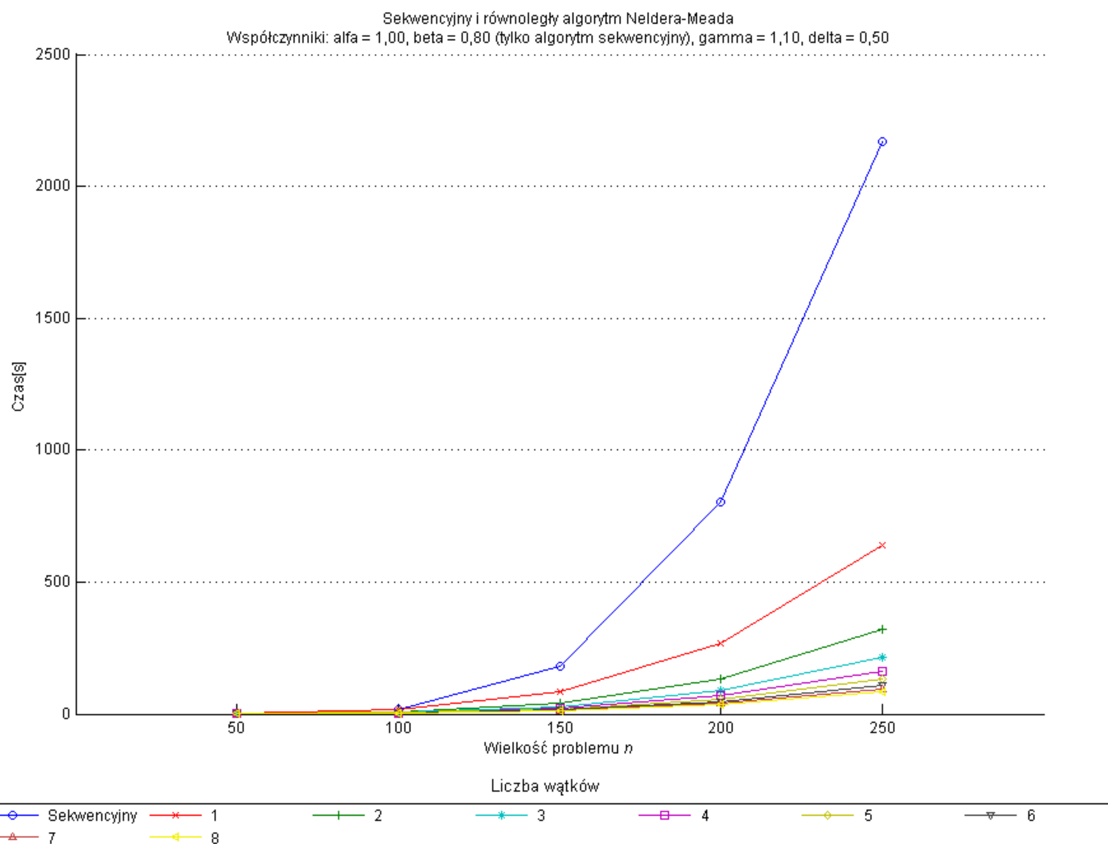
Rys. 3.18 Przyspieszenie zrównoleglonej wersji algorytmu Neldera-Meada dla 2 funkcji testowej. Dla współczynników alfa= 1,00, gamma= 2,00, delta= 0,50



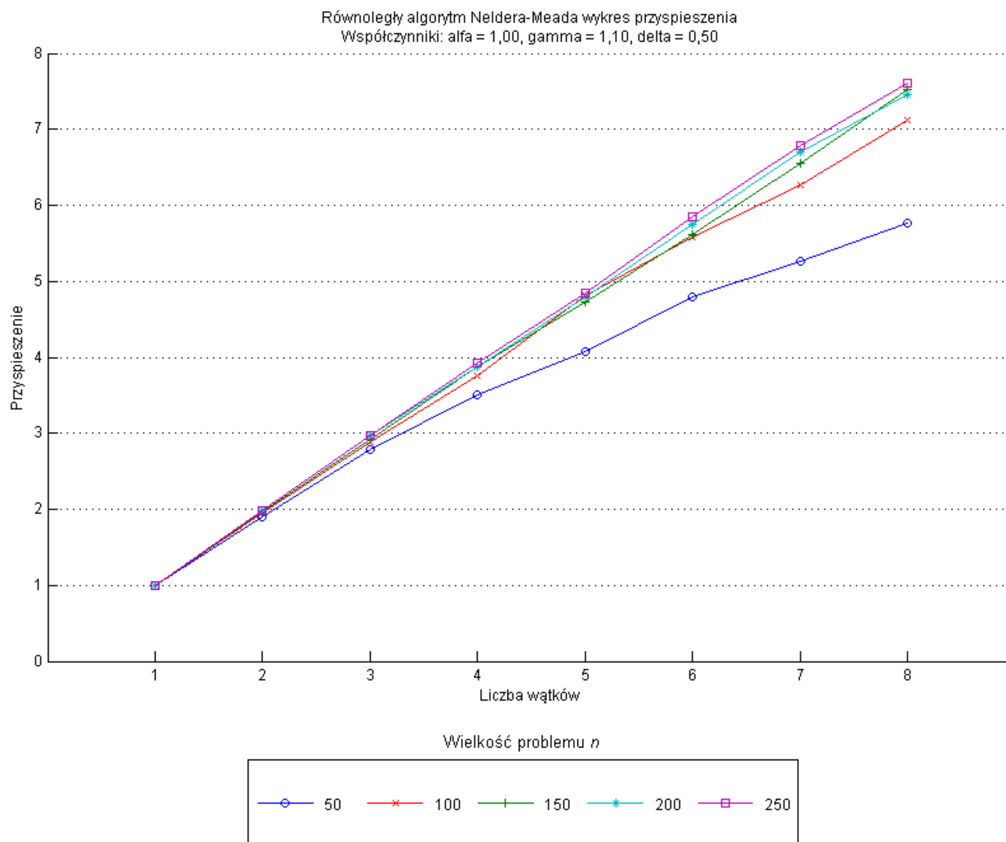
Rys. 3.19 Porównanie czasu obliczeń sekwencyjnego i równoległego algorytmu Neldera-Meada dla 2 funkcji testowej. Dla współczynników alfa= 1,00, beta= 0,80, gamma= 1,20, delta= 0,50



Rys. 3.20 Przyspieszenie zrównoleglonej wersji algorytmu Neldera-Meada dla 2 funkcji testowej. Dla współczynników alfa= 1,00, gamma= 1,20, delta= 0,50



Rys. 3.21 Porównanie czasu obliczeń sekwencyjnego i równoległego algorytmu Neldera-Meada dla 2 funkcji testowej. Dla współczynników alfa= 1,00, beta= 0,80, gamma= 1,10, delta= 0,50



Rys. 3.22 Przyspieszenie zrównoleglonej wersji algorytmu Neldera-Meada dla 2 funkcji testowej. Dla współczynników alfa= 1,00, gamma= 1,10, delta= 0,50

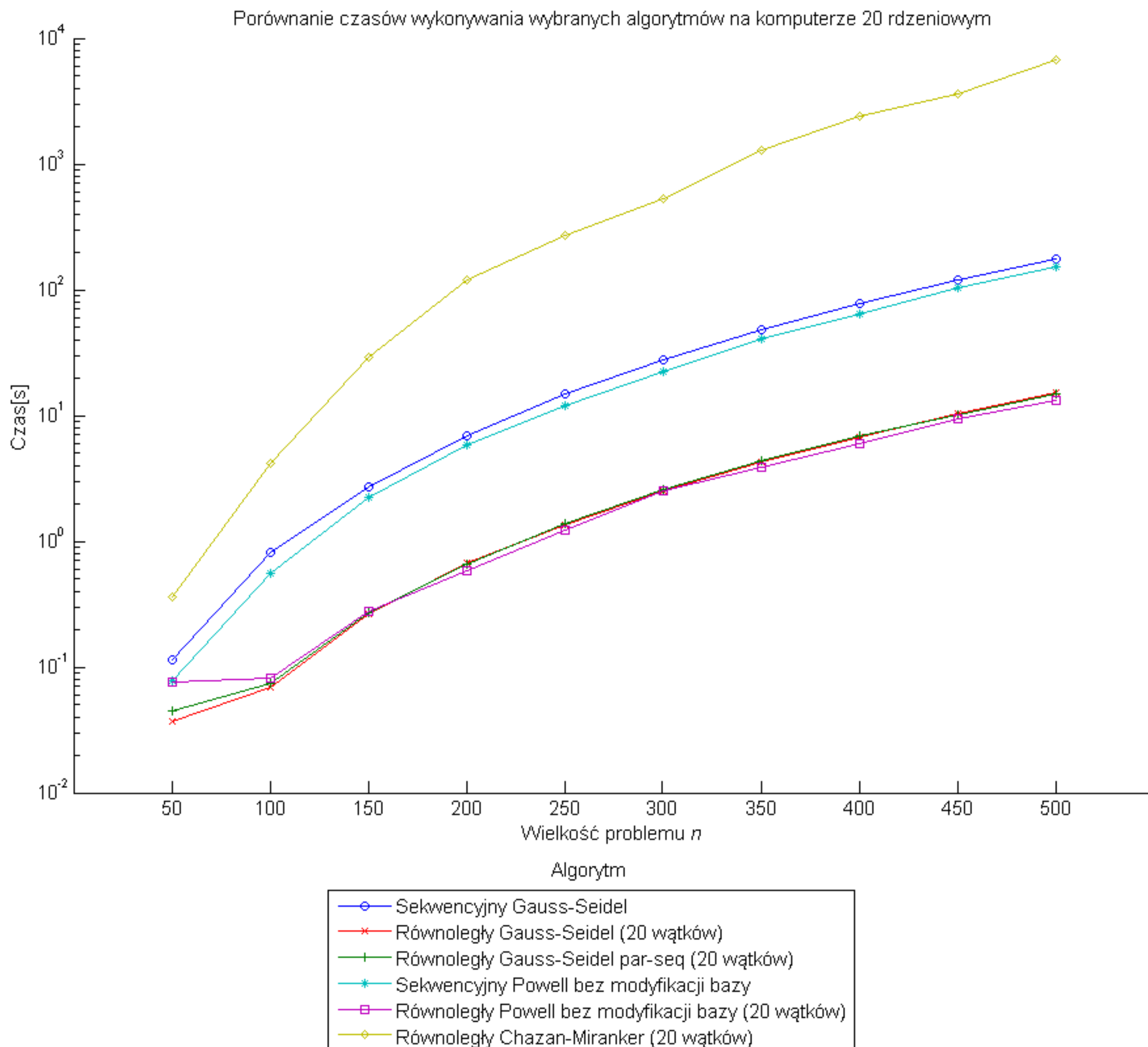
Wyniki przedstawione na wykresach (od Rys. 3.17 do Rys. 3.22) pokazują, że dla przyjętych współczynników algorytm sekwencyjny jest znacznie wolniejszy niż algorytm równoległy. Można również zauważyć, że dobór współczynników ma duży wpływ na czas wykonywania algorytmu sekwencyjnego (Rys. 3.17, 3.19 i 3.21). Dla tej funkcji testowej można również zaobserwować, że równoległa wersja algorytmu Neldera-Meada jest bardziej odporna na dobór współczynników niż sekwencyjny algorytm. Na wykresach (Rys. 3.18, 3.20 i 3.22) można zauważyć, że przyśpieszenia uzyskiwane przez równoległą wersję przekraczają 7 na komputerze 8 rdzeniowym, oprócz wielkości problemu n = 50.

3.6 Porównanie czasów wykonywania wybranych algorytmów

W niniejszym podrozdziale zostaną zaprezentowane porównania czasów obliczeń wybranych algorytmów, które zostały przetestowane na 20 rdzeniowym komputerze. Wielkość problemu n była zależna od wyników przeprowadzonych badań. Wybrano wyniki dla wersji sekwencyjnych oraz równoległych uruchomionych na 20 wątkach. W przypadku algorytmu Chazana-Mirankera jedynie wersja równoległa zostanie przedstawiona, ponieważ nie wymagał

on ingerencji w stukturę algorytmu. Natomiast w przypadku algorytmu Powella z modyfikacją bazy został on pominęty dla dwóch pierwszych funkcji testowych, ponieważ nie dochodziło do modyfikacji bazy kierunków.

Wykres przedstawiający wyniki dla funkcji testowej (3.2):



Rys. 3.23 Porównanie czasów obliczeń dla wybranych algorytmów dla 2 funkcji testowej.

Wyniki zaprezentowane na wykresie Rys. 3.23 pokazują, że dla tej funkcji testowej algorytmy zachowywały się podobnie. Najwolniejszym algorytmem był algorytm Chazana-Mirankera. Sekwencyjny algorytm Gaussa-Seidela oraz sekwencyjny algorytm Powella bez modyfikacji bazy osiągały krótsze czasy obliczeń. Najlepsze czasy obliczeń osiągnęły równoległe wersje algorytmów Gaussa-Seidela, Gauss-Seidla w wersji par-seq oraz algorytm Powella bez modyfikacji bazy uruchomione na 20 wątkach.

Rozdział 4

Optymalizacja dynamiczna

Optymalizacja dynamiczna podobnie do optymalizacji statycznej zajmuje się poszukiwaniem najlepszego (optymalnego) rozwiązania problemu. W problemach optymalizacji dynamicznej, która jest wyznaczaniem sterowania optymalnego, poszukujemy dopuszczalnego i docelowego sterowania jako funkcji czasu t, tak aby przy tym sterowaniu wskaźnik jakości miał wartość minimalną [15, 16, 21, 22],

$$Q\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}, \hat{\boldsymbol{a}}\} = \min Q\{\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{a}\} \tag{4.1}$$

gdzie: - $\hat{\boldsymbol{x}}$ – jest trajektorią optymalną procesu

$\boldsymbol{x}$ – dowolne sterowanie docelowe

$\boldsymbol{u}$ – dowolne sterowanie dopuszczalne

Wskaźnikiem jakości sterowania może być funkcjonał. Maksymalizowany lub minimalizowany funkcjonał traktowany jest jako przyporządkowanie funkcji oraz liczby – wyniku. Funkcjonał może przyjąć następującą postać:

$$Q\{\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{a}\} = f_k(\boldsymbol{x}(t_k), t_k, \boldsymbol{a}) + \int_{t_0}^{t_k} f_0(\boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{a}) \mathrm{d}t \tag{4.2}$$

gdzie: - f_k – funkcja stanu końcowego

f_0– funkcja podcałkowa

Funkcjonał zapisany w ten sposób (4.2) określany jest jako problem Bolzy, natomiast jeżeli $f_k = 0$ to jest to problem Lagrange'a i wskaźnik jakości ma postać tylko całki:

$$Q = \int_{t_0}^{t_k} f_0(\boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{a}) \mathrm{d}t \tag{4.3}$$

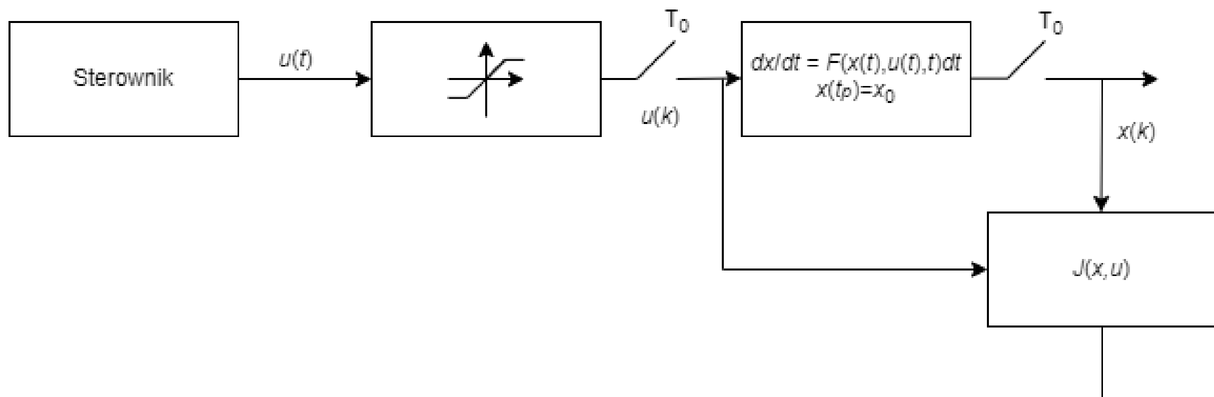
Jeżeli $f_0 = 0$ to jest to problem Mayera, a wskaźnik jakości przyjmuje postać funkcji stanu końcowego:

$$Q = f_k(\boldsymbol{x}(t_k), t_k, \boldsymbol{a}) \tag{4.4}$$

Jednakże funkcjonał może przyjąć również inne postacie.

W zadaniach optymalizacji dynamicznej poszukiwana jest funkcja, natomiast w optymalizacji statycznej poszukiwany był punkt. W szczególnych przypadkach można jednak zastosować metody optymalizacji statycznej do rozwiązywania problemów optymalizacji dynamicznej.

Sterowanie optymalne w optymalizacji dynamicznej[1], dla pewnej klasy problemów, może być wyznaczane za pomocą metod optymalizacji statycznej. Dla rozważanego (Rys. 4.1), dyskretnego, liniowego systemu dynamicznego należy wygenerować ciąg sterujący $\boldsymbol{u} = \{\boldsymbol{u}(kT_0)\}$. Ciąg sterujący minimalizujący wartość funkcji celu dla skończonego horyzontu czasowego sterowania.



Rys. 4.1 Dyskretny liniowy system dynamiczny (wzorowany na notatkach do wykładu[1])

Taki problem można rozwiązać poprzez sprowadzanie zadania do rozwiązywania problemu optymalizacji statycznej. Przykładowe metody rozwiązywania tego rodzaju problemu:

- poszukiwanie ekstremum w przestrzeni sterowań z jednoczesnym rozwiązywaniem równań stanu
- poszukiwanie ekstremum w przestrzeni sterowań i stanu.

W przypadku takich metod należy określić horyzont sterowania T i dokonać aproksymacji przestrzeni sterowań, której reprezentację możemy opisać wzorem:

$$\hat{\boldsymbol{u}}(\boldsymbol{a}, t) = \sum_{i=0}^{M} a_i \, \varphi_i(t) \tag{4.5}$$

[1] Grega Wojciech, METODY OPTYMALIZACJI Notatki do wykładu, Kraków 2001

gdzie:

$\varphi_i(t)$ – funkcje bazowe

$a_i(t)\epsilon R^{m_u}$ – są współczynnikami rozwinięcia

Zadanie optymalizacji dynamicznej:

$$\min J(\boldsymbol{x},\boldsymbol{u}) = min \int_{t_p}^{t_k} f_0\,(\boldsymbol{x}(t),\boldsymbol{u}(t),t)\mathrm{d}t \tag{4.6}$$

$$(\boldsymbol{x},\boldsymbol{u})\epsilon X \times U \tag{4.7}$$

przy przyjętym ograniczeniu w postaci równań stanu zapisanych jako:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = F(\boldsymbol{x}(t),\boldsymbol{u}(t),t) \tag{4.8}$$

$$\boldsymbol{x}(t_p) = \boldsymbol{x_0} \tag{4.9}$$

może zostać zapisane jako:

$$\min \hat{J}(\boldsymbol{a}) = min \int_{t_p}^{t_k} f_0\,(\boldsymbol{x}(\boldsymbol{\hat{u}}(\boldsymbol{a},t)),\boldsymbol{\hat{u}}(\boldsymbol{a},t),t)\mathrm{d}t \tag{4.10}$$

gdzie:

$\boldsymbol{x}(\boldsymbol{\hat{u}}(\boldsymbol{a},t))$ – rozwiązanie układu równań stanu dla danych parametrów a

$$\frac{\mathrm{d}x}{\mathrm{d}t} = F(\boldsymbol{x}(t),\boldsymbol{\hat{u}}(\boldsymbol{a},t),t) \tag{4.11}$$

oraz warunek początkowy zapisany w postaci:

$$\boldsymbol{x}(t_p) = \boldsymbol{x_0} \tag{4.12}$$

O jakości aproksymacji decyduje dobór i liczba funkcji bazowych: $\{\varphi_i(t)\}$, gdzie $i = 1..M$.

Użycie tej metody zostało zaprezentowane za pomocą poniższego przykładu. Metodą optymalizacji statycznej, która została wykorzystana do rozwiązywania problemów, jest zrównoleglona metoda Neldera-Meada. Ta metoda została opisana w rozdziale 3.5. Przyjęte współczynniki dla operacji wykonywanych na sympleksie były następujące alfa= 1,00, gamma= 2,00, delta= 0,50. Obliczenia przeprowadzono na komputerze o specyfikacji:

- AMD RYZEN 7 5800X 3.8 GHz (8 rdzeni)

- 32 GB pamięci RAM
- dysk SSD
- system operacyjny typu Linux.

Przykład:

Dla przykładowego układu opisanego poniższym równaniem stanu:

$$\dot{x} = u, \qquad x(0) = 8, \qquad t\epsilon < 0{,}10 > \tag{4.13}$$

znaleźć takie sterowanie *u,* które przy spełnieniu równia stanu zminimalizuje następujący wskaźnik jakości:

$$J = \int_0^{10} (x^2 + u^2)dt + 2{,}5(x(10) - 2)^2 \tag{4.14}$$

Całkowanie metodą trapezów – po *x*

Całkowanie metodą prostokątów - po *u*

$$J = \int_0^{10} (x^2 + u^2)\mathrm{d}t = J_1 + J_2 = \int_0^{10} x^2\mathrm{d}t + \int_0^{10} u^2\mathrm{d}t \tag{4.15}$$

gdzie: J_1, J_2 - lokalne wskaźniki jakości

$$J_2 = \sum_{k=1}^{kd-1} u^2(k)\,\Delta t, \tag{4.16}$$

gdzie: $\mathrm{d}t = \Delta t$ - krok całkowania

$$J_1 = \left[\underbrace{\left(\sum_{k=2}^{kd-1} x(k)^2 \right)}_{s1} + \underbrace{\frac{1}{2}x^2(1)}_{s2} + \underbrace{\frac{1}{2}x^2(kd)}_{s3} \right] * \mathrm{d}t, \tag{4.17}$$

gdzie: *s*1 - suma pól trapezów (bez wartości brzegowych),

*s*2, *s*3 - wartości brzegowe,

$\mathrm{d}t = \Delta t$ - krok całkowania

$$s_2 = \frac{1}{2}x^2(0) * \Delta t \tag{4.18}$$

$$x(kd) = x(0) + (\sum_{k=1}^{kd-1} u(k)) * \Delta t \tag{4.19}$$

$$s_3 = \frac{1}{2}(x(0) + \sum_{k=1}^{kd-1} u(k) * \Delta t)^2 * \Delta t \tag{4.20}$$

$$x(k) = x(0) + \sum_{i=1}^{k-1} u(i) * \Delta t \tag{4.21}$$

$$s_1 = \sum_{k=2}^{kd-2} x^2(k) * \Delta t = \sum_{k=2}^{kd-1}(x(0) + \sum_{i=1}^{k-1} u(i) * \Delta t)^2 * \Delta t \tag{4.22}$$

$$\dot{x} = u, \qquad x(0) = 8, \qquad t\epsilon < 0{,}10 > \tag{4.23}$$

$$J = \int_0^{10} (x^2 + u^2)\mathrm{d}t + 2.5(x(10) - 2)^2 \tag{4.24}$$

Zależność 4.24 w wersji dyskretnej przyjmuje postać:

$$J = \sum_{k=2}^{kd-1}(x(0) + \sum_{i=1}^{k-1} u(i) * \Delta t)^2 * \Delta t + \frac{1}{2}x^2(0) * \Delta t + \frac{1}{2}\left(x(0) + \sum_{k=1}^{kd-1} u(k) * \Delta t\right)^2 * \Delta t + \sum_{k=1}^{kd-1} u^2(k)\,\Delta t + 2.5(x(10) - 2)^2 \tag{4.25}$$

Dane przykładu:

- Kd =101

- i = 1:100

- dt = 0,01

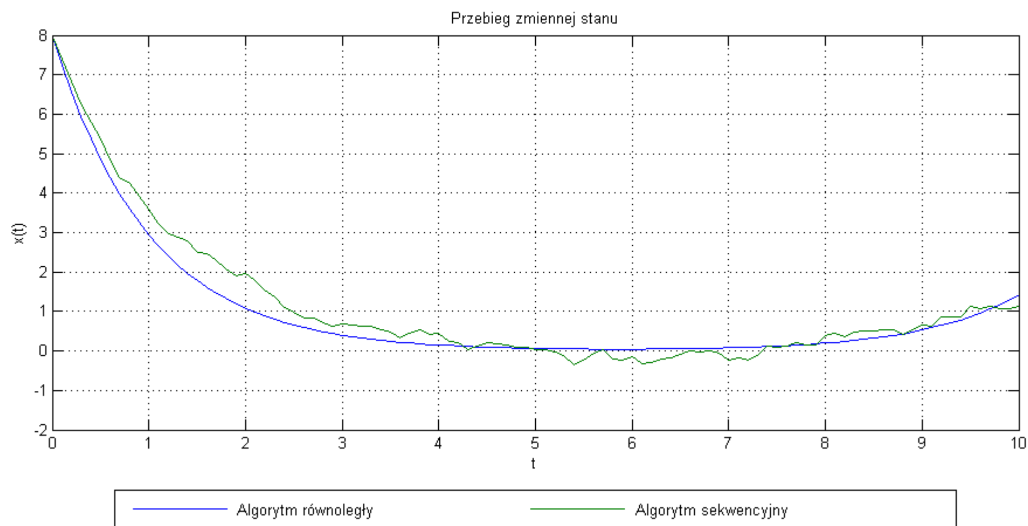
- $t(\min)$ = 0

- $t(max) = 10$

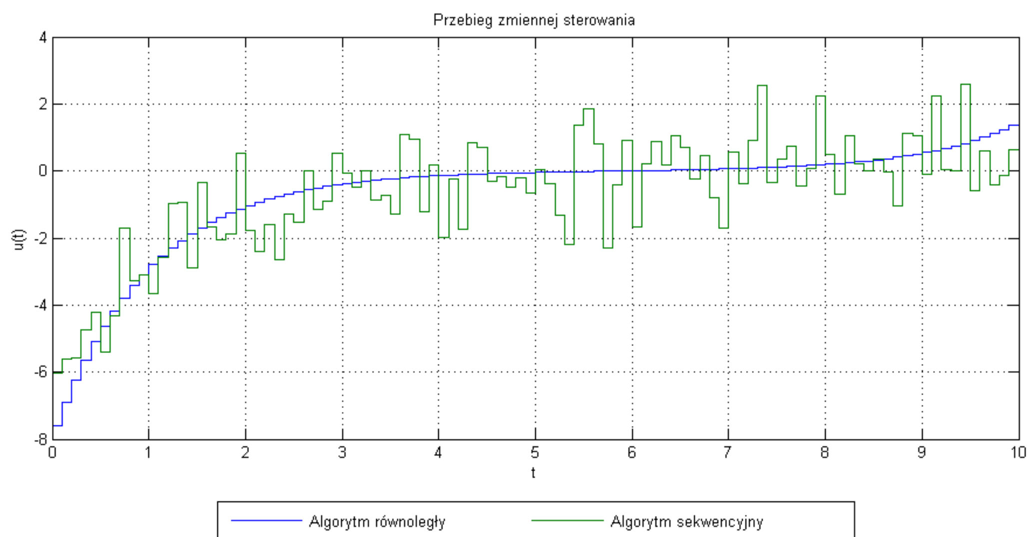
- $J = 66{,}9376$

Przybliżenie początkowe:

- $u(i) - 0.7$



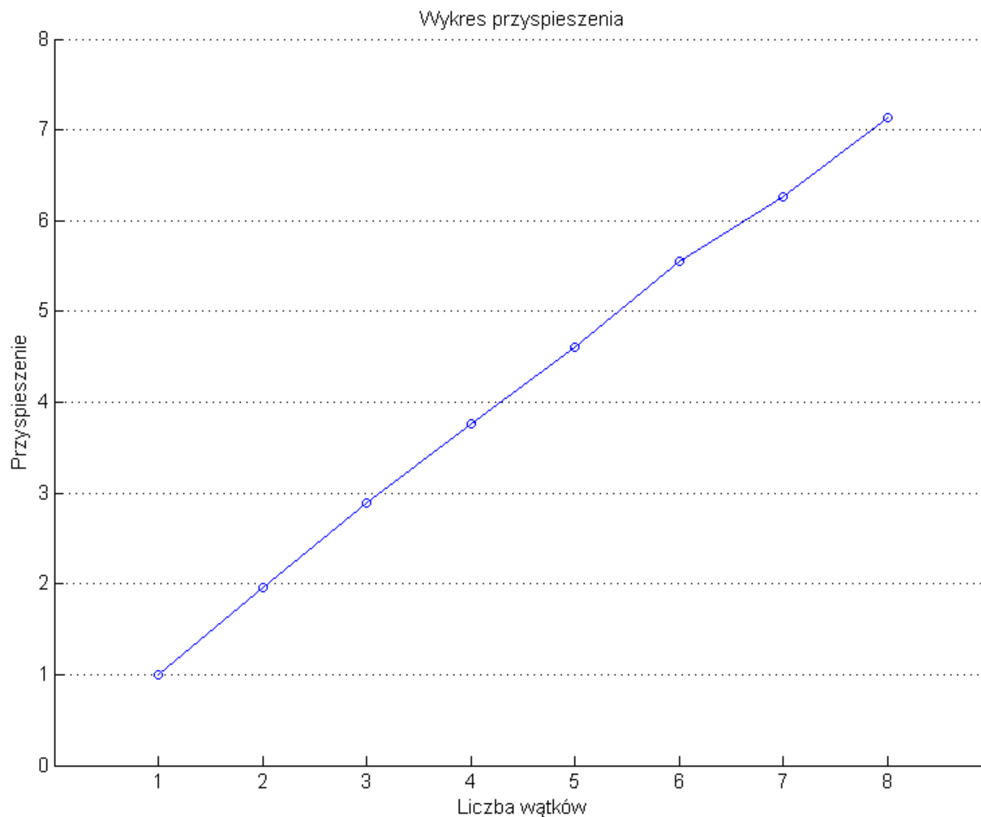
Rys. 4.2 Przebieg zmiennej stanu



Rys. 4.3 Przebieg zmiennej sterowania

Wykresy (Rys. 4.2, Rys. 4.3) przedstawiają przebiegi zmiennej czasu oraz sterowania dla przykładowego zadania (4.14) dla $Kd = 101$, $i = 1{:}100$. Wersja sekwencyjna to funkcja

fminsearch z programu Matlab. Na obu wykresach można zauważyć, że algorytm w wersji sekwencyjnej wykazywał znaczące błędy, które wynikały z samej istoty zastosowanego algorytmu.



Rys. 4.4 Wykres przyśpieszenia dla liczby wątków od 1 do 8

Na wykresach (Rys. 4.2, Rys. 4.3) można zaobserwować znaczące rozbieżności pomiędzy wersją sekwencyjną oraz wersją równoległą. Różnice te mogą wynikać z tego, że w sekwencyjnej wersji algorytm Neldera-Meada może dochodzić do tzw. redukcji wymiaru sympleksu, skutkującej spowolnieniem algorytmu oraz nierzadko jego zapętlaniem się wokół lokalnych ekstremów funkcji. Problem ten generalnie nie występuje w równoległej wersji algorytmu Neldera-Meda zaproponowanego w [38]. Zastosowanie równoległej wersji algorytmu Neldera-Meada oraz zastosowanie odpowiedniej dyskretyzacji, a także dobranie odpowiedniego przybliżenia początkowego dla powyższego przykładu pozwoliło otrzymać prawidłowe przebiegi. Sam algorytm w wersji równoległej uzyskał przyśpieszenie przekraczające 7 dla komputera z 8 wątkowym/rdzeniowym procesorem (Rys. 4.4).

Rozdział 5

Uwagi końcowe

5.1 Wnioski

Zaprezentowane wyniki potwierdzają przyjętą w tej pracy tezę o możliwości poprawy efektywności analizowanych algorytmów optymalizacji statycznej poprzez konstrukcje nowych rozwiązań lepiej dopasowanych do różnych typów architektury równoległej. Dzięki zrównolegleniu tych algorytmów można lepiej wykorzystać dostępne zasoby czy to zwykłego komputera stacjonarnego czy klastra komputerowego. Wyniki potwierdzają również, że dostosowanie klasycznych metod optymalizacji do systemów równoległych pozwala znacząco skrócić czas wykonywania obliczeń, co może prowadzić do poszerzenia zakresu problemów, do których mogą one zostać zaaplikowane w akceptowalnym czasie. Zrównoleglenie może także prowadzić do poprawienia zbieżności algorytmu jak to miało miejsce w przypadku algorytmu Neldera-Meada, którego równoległa wersja algorytmu cechowała się lepszą zbieżnością niż wersja sekwencyjna.

Dla większości przypadków osiągnięto skrócenie czasu wykonywania algorytmu. Tylko w niektórych przypadkach, głównie dla wielkości problemu n równego 50, zanotowano pewne niezadowalające zachowania wynikające z nierównomiernego podziału zadań i relatywnie małej masywności problemu (bardzo krótki czas obliczeń). Z tego względu zostaną one pominięte w dalszej części tego rozdziału i nie będą brane pod uwagę podczas określania uzyskanego zakresu przyśpieszeń.

Zrównoleglony za pomocą OpenMP algorytm Gaussa-Seidela dla dwóch pierwszych funkcji testowych został przetestowany dla wielkości problemu n od 50 do 1000 z krokiem 50, natomiast dla trzeciej funkcji testowej zakres wielkości problemu n zaczynał się od 50 a kończył na 250 również z krokiem 50. Mniejszy zakres dla 3 funkcji testowej związany był z długim czasem wykonywania algorytmu. Dla wszystkich trzech funkcji testowych ten algorytm potrzebował więcej iteracji do odnalezienia minimum niż w wersji sekwencyjnej. Dla 2 pierwszych funkcji testowych wzrost liczby iteracji był niewielki, a sam algorytm sekwencyjny nie potrzebował wielu iteracji do odnalezienia minimum. Natomiast dla testowej funkcji

Rosenbrocka liczba iteracji potrzebna do odnalezienia minimum przez wersję sekwencyjną jest znacząco większa. W badanym zakresie jest to od kilku do kilkunastu tysięcy iteracji, a dla wersji równoległej liczba ta wzrosła około dwukrotnie. Jednakże możliwość prowadzenia obliczeń na wielu wątkach przyniosła znaczące skrócenie czasu wykonywania algorytmu. Dla dwóch pierwszych funkcji testowych krótszy czas wykonywania uzyskano już przy wykorzystaniu 2 wątków. Dla najtrudniejszej funkcji, to jest funkcji Rosenbrocka (3.3), użycie 3 wątków do przeprowadzania obliczeń skutkowało uzyskaniem krótszego czasu wykonywania. Dla większości badanych wielkości problemu n wraz ze wzrostem liczby wątków, na których przeprowadzano obliczenia, czas działania algorytmu ulegał skróceniu. Obliczenia wykonywane były na komputerze 20 rdzeniowym, a więc liczba wątków, na których uruchamiane były testy wynosiła od 1 do 20. Dla 20 wątków uzyskane przyśpieszenie dla pierwszej funkcji mieściło się w zakresie od około 11 do około 14, dla drugiej funkcji od około 13 do około 15,5. Natomiast dla funkcji Rosenborcka przyśpieszenie wynosiło od około 5,5 do około 10. Algorytm Gaussa-Seidela w wersji par-seq (lokalnie sekwencyjny) dla 1 wątku zachowywał się jak algorytm w wersji sekwencyjnej, natomiast wraz ze wzrostem liczby wątków jego działania zbliżało się do algorytmu równoległego. Dla liczby wątków równej wielkości problemu n ta wersja działa tak samo jak wersja równoległa. W związku z tym dla mniejszej liczby wątków ten algorytm potrzebował mniej iteracji, a co za tym idzie był szybszy niż algorytm równoległy. Dla dwóch pierwszych funkcji testowych wraz ze wzrostem liczby wątków różnice w czasach pomiędzy tą wersją, a wersją równoległa szybko się zmniejszały, dla 20 wątków praktycznie nie było różnicy. Dla funkcji Rosenbrocka ta tendencja była znacznie wolniejsza, spowodowane jest to dużą liczbą iteracji potrzebnych do odnalezienia minimum. Dla 20 wątków algorytm par-seq dla pierwszej funkcji testowej uzyskał przyśpieszenia od około 8 do około 13, dla drugiej funkcji testowej od około 10 do około 12, natomiast dla trzeciej funkcji uzyskano przyśpieszenia od około 8 do około 11,5.

Równoległy algorytm z Gaussa-Seidela z modyfikacją bazy (ARm) został przetestowany na 8 rdzeniowym komputerze dla wielkości problemu n od 50 do 500 co 50. Dla dwóch pierwszych funkcji testowych w prawie wszystkich przypadkach potrzebował mniej iteracji do odnalezienia minimum niż algorytm w wersji sekwencyjnej i w wersji równoległej. Natomiast dla testowej funkcji Rosenbrocka ta wersja algorytmu potrzebowała najwięcej iteracji. Uzyskane przyśpieszenia dla 8 wątków dla pierwszej oraz drugiej funkcji testowej mieściły się w zakresie od około 5,5 do około 7, dla trzeciej funkcji zakres ten wynosił od około 1,75 do około 3,3. Algorytm w tej wersji jest najbardziej skomplikowaną i rozbudowaną wersją

algorytmu Gauss-Seidela opisaną w tej pracy. Opiera się on na cyklicznym odświeżaniu bazy kierunków oraz warunkowych modyfikacjach tej bazy. Częstość tych operacji bazuje na eksperymentalnie wyznaczonych współczynnikach, które mogą mieć znaczący wpływ na działanie algorytmu.

Algorytm Powella został przetestowany w kilku wersjach. Pierwszą najbardziej podstawową była wersja bez modyfikacji bazy kierunków, jedynie z przeprowadzaniem dodatkowej minimalizacji wzdłuż nowego kierunku wyznaczanego po wykonaniu wszystkich minimalizacji wzdłuż kierunków znajdujących się w ortogonalnej bazie kierunków. Obliczenia dla wersji sekwencyjnej oraz dla wersji równoległej przeprowadzone były na komputerze 20 rdzeniowym dla wielkości problemu n od 50 do 1000 z krokiem 50. Równoległa wersja algorytmu Powella dla wszystkich funkcji testowych potrzebowała więcej iteracji do odnalezienia minimum niż wersja sekwencyjna. Dwie pierwsze funkcje testowe potrzebowały od kilku do kilkudziesięciu iteracji więcej. Dla testowej funkcji Rosenbrocka ten problem jest najbardziej widoczny, na przykład dla wielkości problemu $n = 1000$ nastąpił wzrost liczby iteracji z około 17 tysięcy dla wersji sekwencyjnej do 72,5 tysiąca dla wersji równoległej. Takie zachowanie przełożyło się negatywnie na czasy działania algorytmu. Dla komputera 20 rdzeniowego dla wielkości problemu $n = 1000$ dla pierwszej funkcji testowej czasy wykonywania algorytmu sekwencyjnego i równoległego były zbliżone dopiero przy obliczeniach prowadzonych na 4 wątkach/rdzeniach. Dla drugiej funkcji testowej różnice nie były aż tak duże i użycie 2 wątków/rdzeni skutkowało krótszym czasem wykonywania zrównoleglonej wersji w porównaniu z wersją sekwencyjną. Natomiast dla funkcji Rosenbrocka potrzeba była aż 7 wątków/rdzeni, aby czas algorytmu równoległego był krótszy niż algorytmu sekwencyjnego. Uzyskane przyśpieszenia dla wielkości problemu $n = 1000$ dla pierwszej funkcji testowej mieściły się w zakresie od około 7 do około 14, dla drugiej funkcji testowej od około 10 do około 14, a dla trzeciej funkcji testowej od około 2 do około 9.

Algorytm Powella z modyfikacją bazy dla dwóch pierwszych funkcji testowych nie różnił się od algorytmu bez modyfikacji bazy, ponieważ nie dochodziło do spełnienia warunku wymaganego do wprowadzania nowego kierunku do bazy. Jedynie dla funkcji Rosenbrocka dochodziło do modyfikacji bazy. Funkcja ta została przetestowana w mniejszym zakresie n (od 50 do 250 z krokiem 50) na komputerze 20 rdzeniowym. Dla równoległej wersji zanotowano znaczący wzrost liczby iteracji względem algorytmu sekwencyjnego, dla $n = 250$ sekwencyjny algorytm potrzebował niecałe 6 tysięcy iteracji, a równoległy algorytm z modyfikacją bazy potrzebował prawie 19,5 tysiąca iteracji. Nawet dla największego przypadku ($n = 250$) czas

wykonywania wersji sekwencyjnej był krótszy niż czas algorytmu równoległego wykonywanego na wszystkich 20 dostępnych wątkach/rdzeniach. Osiągane przyśpieszenia również nie były wysokie – od około 2,5 do około 3,5.

Inną testowaną wersją algorytmu Powella była wersja z przeszukiwaniem wiązką, które miało miejsce na końcu każdej iteracji. Przeszukiwanie wiązką miało zapewnić lepsze wykorzystanie zasobów komputera. Jako, że była to modyfikacja równoległego algorytmu Powella to nie porównano jej z wersją sekwencyjną. Dla wszystkich 3 funkcji testowych obliczenia przeprowadzono w zakresie n od 50 do 250 z krokiem 50 na komputerze 20 rdzeniowym i dla wszystkich funkcji uzyskano znaczące skrócenie czasu rosnące wraz ze wzrostem liczby użytych wątków. Dla 20 wątków dla 1 i 2 funkcji testowej przyśpieszenia mieściły się w zakresie 10 i 14, natomiast dla 3 funkcji testowej zakres był ten większy to jest pomiędzy 6, a 12.

Następnym testowanym algorytmem był algorytm Chazana-Mirankera, który już jest dostosowany do przetwarzania równoległego. Skutkowało to tym, że algorytm sekwencyjny działał dokładnie tak samo jak algorytm równoległy wykonywany na 1 wątku, a jego zrównoleglenie nie wymagało ingerencji w sposób działania. Duża liczba iteracji, jakie były potrzebne do odnalezienia minimum sprawiło, że algorytm potrzebował dużo czasu do odnalezienia minimum, dlatego testy ograniczono do maksymalnej wielkości problemu n równego 500. Testy na komputerze 20 rdzeniowym pokazały, że użycie większej liczby wątków (do 20) znacząco skraca czas wykonywania, a co za tym idzie algorytm osiągnął dobre współczynniki przyśpieszenia. Dla 20 wątków dla pierwszej funkcji testowej przyśpieszenie było w zakresie od około 14 do około 16, dla drugiej funkcji testowej przyśpieszenie mieściło się pomiędzy około 12 i około 16, natomiast dla trzeciej funkcji testowej od około 4 do około 6,5. Ten algorytm lepiej sprawdził się dla trudniejszej testowej funkcji Rosenbrocka niż dla dwóch pierwszych funkcji testowych.

Kolejnym algorytmem, który został przetestowany jest algorytm Nelder-Meada. W swoim działaniu różni się od pozostałych algorytmów, ponieważ nie bazuje na minimalizacjach kierunkowych. Oparty jest o wygenerowanie sympleksu początkowego i jego przekształcanie, aż do momentu spełnienia kryterium stopu. Ta metoda w wersji równoległej zaproponowanej przez Virginię Torczon [7, 38] została przetestowana na 8 rdzeniowym komputerze dla wielkości problemu n od 50 do 250 (z krokiem 50), dla 3 różnych zestawów współczynników

używanych podczas operacji przekształcania sympleksu. Badania przeprowadzono dla 3 eksperymentalnie wyznaczonych zestawów współczynników:

- $\alpha = 1{,}00$, $\beta = 0{,}80$, $\gamma = 2{,}00$, $\delta = 0{,}50$
- $\alpha = 1{,}00$, $\beta = 0{,}80$, $\gamma = 1{,}20$, $\delta = 0{,}50$
- $\alpha = 1{,}00$, $\beta = 0{,}80$, $\gamma = 1{,}10$, $\delta = 0{,}50$.

W algorytmie w wersji równoległej nie występuje operacja redukcji (beta). Te 3 zestawy współczynników nie miały znaczącego wpływu na czas trwania algorytmu równoległego niezależnie od funkcji testowej. Miały natomiast znaczący wpływ na czas wykonywania algorytmu sekwencyjnego dla 2 pierwszych funkcji testowych. Czas potrzebny do odnalezienia minimum przez wersję sekwencyjną został skrócony kilkukrotnie poprzez zastosowanie 2 i 3 zestawu współczynników względem 1 zestawu współczynników. W przypadku 3 funkcji testowej najlepiej wypadł 1 zestaw współczynników, 2 i 3 zestaw pogarszały czas jednak nie były to duże różnice. Dla wszystkich funkcji testowych i wszystkich zestawów współczynników algorytm w wersji równoległej niezależnie od liczby wątków był znacząco szybszy niż wersja równoległa. Przyśpieszenie algorytmu równoległego na komputerze 8 rdzeniowym przy wykorzystaniu 8 wątków dla pierwszej funkcji testowej wynosiło więcej niż 7 niezależenie od zestawu współczynników. Dla drugiej funkcji testowej przyśpieszenie również przekraczało 7 niezależnie od zastosowanych współczynników. Natomiast dla trzeciej funkcji testowej niezależnie od współczynników przyśpieszenie przekraczało 3. Równoległa wersja algorytmu Neldera-Meada, ze względu na odmienny sposób działania niż algorytm sekwencyjny, w przeprowadzonych testach była odporna na redukcję wymiaru sympleksu.

Kolejną testowaną wersją algorytmu Neldera-Meada była wersja równoległo-sekwencyjna, która została przetestowana na 8 rdzeniowym komputerze. Dla dwóch pierwszych funkcji testowych zakres wielkości problemu n wynosił od 50 do 250 z krokiem 50, natomiast dla 3 funkcji testowej zakres ten kończył się na 200. Badania przeprowadzono dla stopnia zrównoleglenia od 2 do 8 dla jednego zestawu współczynników:

- $\alpha = 1{,}00$, $\beta = 0{,}80$, $\gamma = 1{,}10$, $\delta = 0{,}50$.

Dla wszystkich 3 funkcji testowych wersja ta była szybsza od wersji sewkencyjnej, ale w badanych przypadkach była wolniejsza niż wersja równoległa. W jednym przypadku dla stopnia zrównoleglenia 2 dla wielkości problemu n 200 dla 1 funkcji testowej algorytm ten był wolniejszy niż wersja sekwencyjna. Dla stopnia zrównoleglenia 8 dla 8 wątków dla 1 i 2 funkcji

testowej algorytm ten osiągnął przyśpieszenie od około 2,5 do około 3. Natomiast dla 3 funkcji testowej przyśpieszenie wynosiło około 1.

Ze względu na dostępne środowisko testowe obliczenia dla biblioteki MPI zostały przetestowane tylko dla dwóch wybranych algorytmów to jest dla równoległej wersji algorytmu Gaussa-Seidela oraz dla równoległej wersji algorytmu Powella bez modyfikacji bazy. Wszystkie obliczenia przeprowadzono na komputerze 8 rdzeniowym. Zrównoleglony algorytm Gaussa-Seidela dla wszystkich 3 funkcji testowych osiągał przyśpieszenie około 6-7 przy wykorzystaniu 8 wątków. Otrzymane czasy wykonywania algorytmu przy pomocy MPI były zbliżone do czasów uzyskany przy użyciu OpenMP. Dla wymiaru problemu n = 500 dla 2 pierwszych funkcji testowych czas wersji równoległej wykonywanej na 1 wątku lub 1 procesie był gorszy niż czas sekwencyjny, ale wykorzystanie 2 wątków bądź procesów skutkowało czasem krótszym (lepszym). Zastosowanie większej liczby wątków lub procesów w dalszym ciągu skracało czas potrzebny na odnalezienie minimum. W przypadku testowej funkcji Rosenborcka potrzeba było 3 wątków/procesów do uzyskania krótszego czasu niż czas wersji sekwencyjnej.

Równoległy algorytm Powella bez modyfikacji bazy z MPI dla 8 wątków osiągał przyśpieszenie około 6 dla wszystkich 3 funkcji testowych. Tylko dla 1 funkcji testowej dla wymiaru problemu n = 100 przyśpieszenie przekroczyło 5. Również dla tego algorytmu porównano czasy dla bibliotek MPI oraz OpenMP i były one zbliżone. Porównanie czasów wersji sekwencyjnej i zrównoleglonych z OpenMP/MPI dla wymiaru problemu n = 500 pokazuje, że dla 1 funkcji testowej potrzebne były 3 wątki/procesy, aby czas wersji równoległej był krótszy niż czas wersji sekwencyjnej. Dla 2 funkcji testowej wystarczały 2 wątki/procesy, aby uzyskać krótszy czas, natomiast dla 3 funkcji testowej potrzeba było aż 6 wątków/procesów.

Porównanie wybranych sekwencyjnych oraz zrównoleglonych algorytmów optymalizacji statycznej uruchomionych na 20 rdzeniowym komputerze pokazało, że czasy obliczeń są bardzo zależne od minimalizowanej funkcji. Dla 1 i 2 funkcji testowej najwolniejszym algorytmem okazał się algorytm Chazana-Mirankera, który dla założonej dokładności potrzebował znacząco więcej iteracji niż inne porównywane algorytmy. Najlepiej pod względem czasu prezentowały się zrównoleglone algorytmy Powella oraz Gaussa-Seidela uruchomione na wszystkich dostępnych wątkach. Natomiast dla 3 funkcji testowej w zakresie

wielkości problemu n od 50 do 250 najszybszym algorytmem okazał się algorytm Chazana-Mirankera, a najwolniejszym sekwencyjny algorytm Gaussa-Seidela.

Obliczenia na karcie GPU z użyciem biblioteki CUDA zostały przeprowadzone na komputerze wyposażonym w procesor Intel Core i5-3570K (4 rdzenie) oraz kartę graficzną NVidia GTX 660 Ti dla wielkości problemu od $n = 50$ do 1000 z krokiem 50. Ze względu na użycie w większości algorytmów algorytmu Brenta, jako metody poszukiwania minimum w jednym kierunku zdecydowano się przeprowadzić jedynie wybrane obliczenia na karcie. Algorytm Brenta działa w oparciu o pętle oraz instrukcje warunkowe, co czyni go złożonym algorytmem, a taka budowa nie jest dopasowana do obliczeń na karcie GPU. Obliczenia do porównania czasu trwania przeprowadzono z użyciem biblioteki OpenMP na tym samym komputerze. Czas wykonywania algorytmu na karcie GPU był lepszy od czasu uzyskanego przez algorytm uruchomiony na 4 wątkach procesora dopiero od wielkości problemu n około 750. W przypadku użycia mniejszej liczby wątków (CPU) karta GPU uzyskiwała lepszy czas dla mniejszych wartości n. Dla wielkości problemu n do około 100 czas działania algorytmu na karcie GPU był wolniejszy niż w zrównoleglonej wersji niezależnie od wykorzystanej liczby wątków procesora.

Na karcie graficznej przeprowadzono dodatkowe obliczenia wykorzystując algorytm mnożenia macierzy. Czasy porównano z czasami wykonywania algorytmu równoległego na 4 rdzeniowym procesorze przy użyciu biblioteki OpenMP. Algorytm mnożenia macierzy w celu zrównoleglenia nie wymaga ingerencji w sposób działania, dlatego zrównoleglona wersja tej metody uruchomiona na 1 wątku działa tak samo jak algorytm sekwencyjny. Przeprowadzono badania dla dwóch wielkości macierzy 2000 x 2000 i 3000 x 3000. W pierwszym przypadku czas wykonywania algorytmu na karcie graficznej był około 113 razy szybszy niż algorytm działający na 1 wątku. Natomiast porównując najlepszy czas uzyskany na procesorze to jest dla 4 wątków, czas wykonywania na karcie graficznej był około 28 krótszy. Przyśpieszenia uzyskane przez procesor były bardzo dobre. Dla 4 wątków uzyskano przyśpieszenie 4 krotne. Dla większych macierzy karta graficzna również uzyskiwała znacznie lepsze czasy niż procesor niezależnie od liczby użytych wątków. Czas na GPU był prawie 105 razy krótszy niż wykonywania algorytmu na procesorze przez 1 wątek, a także czas GPU był około 26 razy krótszy niż najlepszy czas na procesorze (4 wątki). Również w tym przypadku przyśpieszenie był liniowe lub bardzo zbliżone do takiego. Przeprowadzone dodatkowe obliczenia na karcie graficznej dotyczące mnożenia macierzy pokazują, że odpowiednio dobrany algorytm potrafi

działać znacznie wydajniej na karcie graficznej. W obu badanych przypadkach karta graficzna była znacznie szybsza niż procesor, nawet w przypadku użycia zrównoleglonego algorytmu.

Efektywność zrównoleglenia nie jest jednakowa dla wszystkich algorytmów. Na jej poziom wpływ mają architektura, do której został dostosowany algorytm, oraz budowa algorytmu. Przykładem architektury, która ma wpływ na czas obliczeń, jest klaster wykorzystujący komunikację pomiędzy węzłami. Komunikacja w zależności od wielkości przesyłanych danych, liczby przesyłanych komunikatów oraz ich częstości, a także synchronizacji (jeżeli jest wymagana) może w różny sposób wpłynąć na czas wykonywania algorytmu. Wiele algorytmów posiada sekwencyjną budowę, która może mieć negatywny wpływ na efektywność zrównoleglenia. Przykładem negatywnego wpływu budowy algorytmu na czas obliczeń jest metoda Powella, która wykorzystuje nowy kierunek, wzdłuż którego przeprowadzana jest dodatkowa optymalizacja. W tej pracy ten kierunek wykonywany jest sekwencyjne to znaczy, że inne jednostki obliczeniowe czekają, aż optymalizacja względem niego zostanie zakończona. Sekwencyjne części kodu, które nie zostają zrównoleglone ograniczają maksymalny poziom zrównoleglenia (prawo Amdahla), a co z tym idzie mają negatywny wpływ na czas obliczeń – kod sekwencyjny wykonuje się tak samo niezależnie od liczby jednostek obliczeniowych (jego czas zależny jest od parametrów komputera, na którym przeprowadzane są obliczenia).

W przypadku użycia biblioteki OpenMP otrzymujemy kod programu, który można uruchomić na procesorach wielordzeniowych na 1 lub wielu wątkach. Natomiast użycie protokołu MPI pozwala na napisanie programu mogącego działać na wielu połączonych ze sobą komputerach (przez sieć). Takie programy opierają się o przesyłanie komunikatów pomiędzy procesami i pamięć rozproszoną, dzięki temu programy są skalowalne. Należy pamiętać, że komunikacja pomiędzy procesami, zależnie od rodzaju połączenia i topologii sieci, wpłynie na czas obliczeń. Zrównoleglenie za pomocą technologii CUDA potrafi przynieść znaczące skrócenie czasu. Jednakże wymaga znajomości architektury oraz odpowiednio zrównoleglonego/dobranego algorytmu. Zastosowanie tej technologii może wymagać znacząco większych modyfikacji algorytmu, który ma zostać zrównoleglony.

5.2 Oryginalne wyniki pracy naukowej

Niniejsza praca zakładała opracowanie równoległych wersji wybranych algorytmów optymalizacji statycznej dla różnych systemów równoległych oraz przeprowadzenie badań efektywności zrównoleglenia tych metod, w oparciu o tzw. współczynnik przyśpieszenia obliczeń dla wybranych funkcji testowych. Kolejnym celem było opracowanie nowych konstrukcji konkurencyjnych pod względem efektywności w stosunku do standardowych algorytmów. Końcowym celem było zastosowanie wybranych algorytmów do rozwiązania przykładowego problemu optymalizacji. Poniżej przedstawiono podsumowanie wyników mających wkład w stan wiedzy:

- Opracowano równoległe implementacje wybranych algorytmów optymalizacji statycznej dla różnych systemów równoległych to jest klastry obliczeniowe, procesory wielordzeniowe oraz procesory GPGPU
- Poszukiwano najbardziej efektywnych równoległych implementacji badanych algorytmów w wykorzystanych systemach równoległych.
- Przeprowadzono analizę możliwości, sposobów oraz efektywności rozwiązywania wielowymiarowych problemów optymalizacji statycznej w rozważanych systemach równoległych.
- Zaimplementowano równoległy algorytm Gauss-Seidela z modyfikacją bazy, który przy odpowiednim doborze współczynników (alfa, odświeżania) dla dwóch pierwszych badanych funkcji w badanych zakresach potrzebował mniej iteracji niż sekwencyjna i równoległa wersja algorytmu Gaussa-Seidela w badanych zakresach wielkości problemu n.
- Zaimplementowano równoległy algorytm Powella z przeszukiwaniem wiązką, który lepiej wykorzystywał dostępne zasoby, ale w większości przypadków potrzebował większej liczby iteracji do odnalezienia minimum niż równoległy algorytm Powella bez i z modyfikacją bazy.
- Przeprowadzono badania wydajności dla wybranych algorytmów optymalizacji statycznej w wersjach zarówno sekwencyjnych jak i równoległych dla dużych wymiarów problemu n dochodzących do 1000 zmiennych.

5.3 Kierunki dalszych badań

Niniejsza praca skupia się tylko wokół wybranych bezgradientowych algorytmów optymalizacji statycznej i wybranych funkcji testowych, a także skupia się na wykorzystaniu tylko trzech popularnych bibliotek programowania równoległego, jakimi są OpenMP, MPI oraz CUDA. Nie wyczerpuje to ani możliwości implementacji innych algorytmów, ani ich zastosowań do innych (trudniejszych) funkcji celu, a także innych przypadków użycia lub wykorzystania innych bibliotek wykorzystywanych do zrównoleglenia algorytmów. Podczas opracowywania poniżej pracy pojawiły się różne problemy między innymi:

- Kontynuacja badań symulacyjnych zrównoleglonych wersji algorytmów przy wykorzystaniu szerszej gamy zadań optymalizacyjnych.
- Przeprowadzenie badań dla innych metod optymalizacji statycznej np. metod gradientowych, newtonowskich.
- Zrównoleglenie pozostałych algorytmów w środowisku CUDA oraz przeprowadzenie porównania z innymi architekturami równoległymi.
- Implementacja równoległych algorytmów w tej pracy opiera się głównie na równoległym przeprowadzeniu optymalizacji kierunkowych, jednakże zasadnym wydaje się być zrównoleglenie wewnątrz pojedynczej minimalizacji kierunkowej. Takie podejście mogłoby sprawdzić się między innymi w algorytmie Powell, w którym to minimalizacja wzdłuż nowego kierunku przeprowadzana jest sekwencyjne. Nawet niewielkie przyśpieszenie miałoby pozytywny wpływ na czas obliczeń.
- Połączenie różnych bibliotek programowania równoległego w celu lepszego wykorzystania zasobów np. połączenie OpenMP i MPI – MPI mogłoby być użyte tylko do przesyłania danych pomiędzy węzłami, a do podziału zdań pomiędzy procesory/rdzenie węzła można by wykorzystać OpenMP. Takie podejście zmniejszyłoby narzut na komunikację pomiędzy procesami, a węzły dokonywałyby obliczeń przy użyciu wątków.
- Zastosowanie opracowanych metod i algorytmów równoległych do rozwiązania innych problemów.

Bibliografia

[1] BANDLER J.W., BIERNACKI R.M., SHAO HUA CHEN, HEMMERS R.H., MADSEN K: Electromagnetic optimization exploiting aggressive space mapping, IEEE Transactions on Microwave Theory and Techniques. 43 (12): 2874–2882, doi:10.1109/22.475649, 1995

[2] BANDLER J.W., BIERNACKI R.M.; CHEN SH., GROBELNY P.A., HEMMERS R.H.: Space mapping technique for electromagnetic optimization. IEEE Transactions on Microwave Theory and Techniques, 42 (12): 2536–2544, 1994, doi:10.1109/22.339794.

[3] BISHNU PRASAD DE., KAR R., MANDAL D., GHOSHAL S.P.: Optimal selection of components value for analog active filter design using simplex particle swarm optimization, International Journal of Machine Learning and Cybernetics. 6: 621–636. 2014. doi:10.1007/s13042-014-0299-0. ISSN 1868-8071. S2CID 13071135.

[4] CERVANTES-GONZÁLEZ J. C., RAYAS-SÁNCHEZ J. E., LÓPEZ C. A., CAMACHO-PÉREZ J R., BRITO-BRITO Z., CHÁVEZ-HURTADO J. L.: Space mapping optimization of handset antennas considering EM effects of mobile phone components and human body, International Journal of RF and Microwave Computer-Aided Engineering. 26: 121–128, doi:10.1002/mmce.20945, 2016.

[5] CSISZAR S.: Optimization Algorithms (survey and analysis), 2007 International Symposium on Logistics and Industrial Informatics, 2007, pp. 185-188, doi: 10.1109/LINDI.2007.4343536.

[6] CZECH Z.: Wprowadzenie do obliczeń równoległych. Wydanie 2, PWN, Warszawa 2020.

[7] DENNIS J.E., TORCZON V.: Direct Search Methods on Parallel Machines, SIAM J. Optimization, vol.1, No.4, November 1991, pp.448 – 474.

[8] DIEWERT E.D.: Cost functions, The New Palgrave Dictionary of Economics, 2nd Edition, 2008.

[9] DORFMAN R.: An Economic Interpretation of Optimal Control Theory. American Economic Review, 1969, 59 (5): 817–831. JSTOR 1810679.

[10] DU D. Z., PARDALOS P. M., WU W.: History of Optimization. In Floudas, C., Pardalos P. (eds.). Encyclopedia of Optimization. Boston: Springer, 2008, pp. 1538–1542.

[11] FREEMAN T. L., PHILLIPS C.: Parallel Numerical Algorithms, Prentice Hall, 1992.

[12] FRIEDRICH N.: Space mapping outpaces EM optimization in handset-antenna design, microwaves&rf, Aug. 30, 2013.

[13] HEGAZY T.: Optimization of Resource Allocation and Leveling Using Genetic Algorithms, Journal of Construction Engineering and Management. 125: 167–175. doi:10.1061/(ASCE)0733-9364(1999)125:3(167), 1999.

[14] HERTY M., KLAR A.: Modeling, Simulation, and Optimization of Traffic Flow Networks. SIAM Journal on Scientific Computing. 25: 1066–1087. doi:10.1137/S106482750241459X. 2003

[15] KACZOREK T.: Podstawy teorii sterowania, WNT 2005.

[16] KACZOREK T.: Teoria sterowania, PWN Warszawa 1981.

[17] KALICZYŃSKA M., SADECKI J.: „Obliczenia równoległe – klastry obliczeniowe", Zeszyty Naukowe PO, s. Elektryka, z.57, 2006, s. 101 - 114.

[18] KARBOWSKI A., NIEWIADOMSKA-SZYNKIEWICZ E.: Programowanie Równoległe i Rozproszone, Oficyna Wydawnicza PW, 2009.

[19] KOOHI I., GROZA V. Z.: Optimizing Particle Swarm Optimization algorithm, 2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE), 2014, pp. 1-5, doi: 10.1109/CCECE.2014.6901057.

[20] KOZIEL S., BANDLER J.W.: Space Mapping With Multiple Coarse Models for Optimization of Microwave Components. IEEE Microwave and Wireless Components Letters. 18 (1): 1–3. doi:10.1109/LMWC.2007.911969. S2CID 11086218. 2008.

[21] KRYSZEWSKI W.: Podstawy Teorii Sterowania Optymalnego, http://www-users.mat.umk.pl/~wkrysz/attachments/sterowanie.pdf, 10.12.2018.

[22] KRYSZEWSKI W.: Sterowanie Optymalne. Wykład monograficzny. Politechnika Łódzka, Łódź 2014.

[23] KUMAR, V., GUPTA, A.: Analyzing Scalability of Parallel Algorithms and Architectures, Tech. Rep. TR-91-18, Comput. Sci. Dept., Univ. of Minnesota, Minneapolis, MN, (June, 1991).

[24] LATIFF N. M. A., TSIMENIDIS C. C., SHARIF B. S.: Performance Comparison of Optimization Algorithms for Clustering in Wireless Sensor Networks, 2007 IEEE International Conference on Mobile Adhoc and Sensor Systems, 2007, pp. 1-4, doi: 10.1109/MOBHOC.2007.4428638.

[25] LEWIS A.: Parallel Optimization Algorithms for Continuous, Non-linear Numerical Simulation, Doctoral thesis, 2004, Griffith Uni-versity, Brisbane, Australia.

[26] MACHACZEK M.: Klastry obliczeniowe - obliczenia równoległe, Zeszyt Naukowy nr 362, Elektryka z. 74, ISSN 1429-1533; ISBN 978-83-65235-55-8,Zeszyt poświęcony studiom doktoranckim na Wydziale Elektrotechniki, Automatyki i Informatyki, Opole, 2016.

[27] MACHACZEK M., SADECKI J.: Analiza efektywności wybranych równoległych realizacji algorytmu Neldera-Meada w środowisku komputerów wielordzeniowych, STUDIA

I MONOGRAFIE z. 434, Współczesne problemy w zakresie inżynierii biomedycznej i neuronauk, Oficyna Wydawnicza Politechnika Opolska, Opole 2016, s. 95-106.

[28] MENDES P., KELL D.: Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. Bioinformatics. 14: 869–883. doi:10.1093/bioinformatics/14.10.869. 1998.

[29] NELDER J. A., MEAD R.: A Simplex Method for Function Minimization, Computer Journal, vol. 7, 1965, pp. 308–313.

[30] PACHECO P. S.: An Introduction to Parallel Programming, Morgan Kaufmann 2011.

[31] PIRYONESI S. M., NASSERI M., RAMEZANI A.: Resource leveling in construction projects with activity splitting and resource constraints: a simulated annealing optimization, Canadian Journal of Civil Engineering. 46: 81–86. doi:10.1139/cjce-2017-0670. 2018.

[32] PIRYONESI S. M., TAVAKOLAN M.: A mathematical programming model for solving cost-safety optimization (CSO) problems in the maintenance of structures, KSCE Journal of Civil Engineering. 21 (6): 2226–2234. doi:10.1007/s12205-017-0531-z. S2CID 113616284. 2017

[33] PROGRAMOWANIE RÓWNOLEGŁE, www.icm.edu.pl/kdm/Programowanie_r%C3%B3wnoleg%C5%82e, 26 października 2013.

[34] ROTEMBERG J., WOODFORD M.: An Optimization-based Econometric Framework for the Evaluation of Monetary Policy, 1997, NBER Macroeconomics Annual. 12: 297–346. doi:10.2307/3585236. JSTOR 3585236.

[35] SADECKI J.: Algorytmy równoległe optymalizacji i badanie ich efektywności; systemy równoległe z rozproszoną pamięcią. Oficyna Wydawnicza Politechniki Opolskiej, Opole 2001.

[36] SHENG T., CHENG QINGSHA S., ZHANG Y., BANDLER J. W., NIKOLOVA N. K.: Space Mapping Optimization of Handset Antennas Exploiting Thin-Wire Models. IEEE Transactions on Antennas and Propagation. 61 (7): 3797–3807, doi:10.1109/TAP.2013.2254695, 2013.

[37] THUY D. VO; PAUL LEE W.N.; BERNHARD O. PALSSON: Systems analysis of energy metabolism elucidates the affected respiratory chain complex in Leigh's syndrome. Molecular Genetics and Metabolism. 91 (1): 15–22. doi:10.1016/j.ymgme.2007.01.012. 2007.

[38] TORCZON V.: On the Convergence of the Multidirectional Search Al-gorithm, SIAM J. Optimization, vol.1, No.1, February 1991, pp.123-145.

[39] WANG RS., WANG Y,. ZHANG XS., CHEN L.: Inferring transcriptional regulatory networks from high-throughput data. Bioinformatics. 23 (22): 3056–3064. doi:10.1093/bioinformatics/btm465. 2007.

[40] WANG Y., JOSHI T., ZHANG XS., XU D., CHEN L.: Inferring gene regulatory networks from multiple microarray datasets. Bioinformatics. 22: 2413–2420. doi:10.1093/bioinformatics/btl396. 2006.

[41] XU L., Wang X.: Comparison of Two Optimization Algorithms for Focused Microwave Breast Cancer Hyperthermia, 2018 International Applied Computational Electromagnetics Society Symposium - China (ACES), 2018, pp. 1-2, doi: 10.23919/ACESS.2018.8669197.

[42] YIN Y., LIAO B., LI S.: A New Optimization Algorithm and Its Comparison on Traditional Optimization Algorithms, 2019 Chinese Control Conference (CCC), 2019, pp. 2698-2701, doi: 10.23919/ChiCC.2019.8866517.