Analiza efektywności wybranych równoległych implementacji algorytmu Gaussa-Seidela

Marek Machaczek, Jan Sadecki
Instytut Automatyki i Informatyki, Wydział Elektrotechniki, Automatyki i Informatyki, Politechnika Opolska, 45-272 Opole, ul. Sosnkowskiego 31

Streszczenie: W artykule przedstawiono analizę porównawczą dotyczącą badania efektywności kilku równoległych implementacji algorytmu Gaussa-Seidela. Analizowany w artykule algorytm pozwala na osiągnięcie dosyć dobrych pod względem szybkości zbieżności oraz wartości współczynnika przyspieszenia obliczeń wyników w porównaniu do standardowej sekwencyjnej oraz równoległej implementacji metody Gaussa-Seidela. Obliczenia praktyczne przeprowadzono w środowisku procesorów wielordzeniowych oraz w środowisku klastrów obliczeniowych.

Słowa kluczowe: algorytmy optymalizacji, obliczenia równoległe, równoległe algorytmy optymalizacji

1. Wprowadzenie

Problemy optymalizacji występują w bardzo wielu dziedzinach współczesnej nauki oraz techniki. Pojawiają się one wszędzie tam, gdzie poszukuje się najlepszych, z punktu widzenia przyjętych kryteriów, rozwiązań. W teorii i praktyce obliczeń optymalizacyjnych znanych jest wiele bardzo różnorodnych metod i algorytmów, często silnie powiązanych z rodzajem zadań i problemów, do których rozwiązania są one wykorzystywane. Dobór właściwego do rozważanego problemu algorytmu zależny jest również od ilości oraz rodzaju dostępnej informacji o przyjętej funkcji celu. W przypadku prostszych algorytmów wystarczające może być zagwarantowanie możliwości wyznaczenia jedynie wartości funkcji celu, podczas, gdy zastosowanie bardziej zaawansowanych metod obliczeniowych może wymagać znajomości gradientu, czy też nawet drugich pochodnych funkcji celu [3, 4]. W przypadku złożonych zadań optymalizacji już samo wyznaczenie wartości funkcji celu może wymagać przeprowadzenia bardzo czasochłonnych obliczeń, wynikających ze złożoności modeli wykorzystywanych do opisu optymalizowanych procesów i zjawisk rzeczywistych. Dodatkowym czynnikiem zwiększającym czasochłonność procedur optymalizacyjnych może być wysoki wymiar rozwiązywanych problemów, co przy stosowaniu nawet najbardziej efektywnych algorytmów optymalizacji może stanowić poważne utrudnienie w procesie ich rozwiązywania.

Bardzo duży potencjał w zakresie znacznego ograniczenia niedogodności wynikających z dużej czasochłonności obliczeń optymalizacyjnych tkwi w możliwości zastosowania do ich realizacji tak programowych, jak i sprzętowych środków umożliwiających równoległe przetwarzania danych [5, 6, 11].

Rozważania przedstawione w niniejszej pracy dotyczą generalnie zadania programowania nieliniowego (ZPN), sformułowanego w punkcie 2 zależnością (1).

Większość z podstawowych algorytmów obliczeniowych optymalizacji ma strukturę wybitnie sekwencyjną, stąd też bezpośrednie ich wykorzystanie do obliczeń w systemach równoległych może okazać się dosyć trudne.

Jest tak np. w odniesieniu do wielu metod i algorytmów programowania nieliniowego, opartych na wykonywaniu sekwencji elementarnych zadań optymalizacji kierunkowych realizowanych dla właściwych dla danych metod sposobów określania kierunków poszukiwań.

Z uwagi na naturalne ograniczenia wielu środowisk przetwarzania równoległego, związane z niekorzystnym wpływem procesu wymiany danych (np. klastry obliczeniowe) lub częstego odwołania do wspólnych danych (np. systemy z pamięcią wspólną) na globalną efektywność praktycznych implementacji algorytmów równoległych, powinno się zmierzać do tego, aby granulacja zadań realizowanych równolegle nie była zbyt drobna. Stąd też przyjęte w niniejszej pracy założenie aby, w odniesieniu do równoległych implementacji rozważanych algorytmów optymalizacji, zadanie optymalizacji kierunkowej było najmniejszą możliwą do wyszczególnienia porcją zadania, wydaje się być uzasadnione.

W implementacjach sekwencyjnych rozważanych algorytmów optymalizacji generowany jest zazwyczaj ściśle sekwencyjny ciąg przybliżeń (rozwiązań) problemu, zbieżny (przy zapewnieniu zbieżności wykorzystywanej metody) do rozwiązania optymal-

Autor korespondujący:
Jan Sadecki, j.sadecki@po.opole.pl

Artykuł recenzowany
nadesłany 22.12.2014 r., przyjęty do druku 23.01.2015 r.

nego. W przypadku równoległej implementacji tych algorytmów otrzymuje się z reguły (na każdym jego etapie) rozproszoną grupę punktów, na bazie których należy określić jeden punkt lub grupę punktów (przybliżeń rozwiązania optymalnego) stanowiących punkt początkowy dla kolejnego etapu algorytmu. Wspomniana wyżej grupa punktów może być otrzymana w wyniku równolegle (jednocześnie) realizowanych wielu zadań optymalizacji kierunkowej, „startujących" z tego samego lub z wielu różnych punktów początkowych, realizowanych przy tym względem jednego, tego samego kierunku poszukiwań (wiązka równoległa) lub wielu (wiązka zbieżna lub rozbieżna) różnych kierunków.

Spostrzeżenie to może być podstawą sformułowania wielu różnych równoległych implementacji tego samego algorytmu sekwencyjnego, przy czym tak sformułowane algorytmy mogą charakteryzować się różnymi warunkami zbieżności oraz różną szybkością zbieżności w porównaniu do wyjściowej, sekwencyjnej ich wersji, jednak ich bezsprzeczną zaletą jest to, że mogą być realizowane w sposób równoległy, a więc pozwalają na wykorzystanie trudnego do przecenienia potencjału, jaki tkwi w równoległych systemach obliczeniowych [11].

Może się też zdarzyć, że opracowana dla danego algorytmu sekwencyjnego wersja równoległa będzie charakteryzowała się zdecydowanie lepszymi właściwościami w porównaniu do jej odpowiednika sekwencyjnego, jak to jest np. w przypadku równoległej wersji algorytmu Neldera-Meada [2, 11].

Algorytm sympleksu Neldera-Meada jest w miarę popularnym i, jak wskazuje literatura przedmiotu (np. pozycja [8, 11]), dosyć chętnie stosowanym algorytmem w obliczeniach praktycznych, tym bardziej że jego implementacja dostępna jest w środowisku MATLAB. Jednakże jego zastosowanie w odniesieniu do realizacji sekwencyjnej może prowadzić do tak zwanej redukcji sympleksu, co skutkuje praktycznie niemożnością wyznaczenia rozwiązania optymalnego bez zastosowania procedury „odświeżania" sympleksu. Ponadto, jak pokazują badania zaprezentowane w pracy [11], dla rozważanych tam przykładów, zastosowanie algorytmu Neldera-Meada daje gorsze wyniki odnośnie czasochłonności obliczeń, (zarówno w implementacji sekwencyjnej, jak i równoległej) w porównaniu do omawianego w niniejszym artykule algorytmu Gaussa-Seidela z dynamiczną modyfikacją bazy kierunków poszukiwań.

Możliwość równoległego prowadzenia obliczeń (np. jednoczesnego poszukiwania ekstremum w różnych kierunkach generowanych według różnych algorytmów) stwarza również realne podstawy poprawy niezawodności stosowanych obecnie algorytmów, co ma znaczenie w odniesieniu do trudnych (np. źle uwarunkowanych) zadań optymalizacji. Straty czasu spowodowane stosowaniem wielu kierunków poszukiwań są tutaj rekompensowane przyspieszeniem wynikającym z równoległego sposobu przetwarzania danych. Może się też zdarzyć, że opracowana dla danego algorytmu sekwencyjnego wersja równoległa będzie charakteryzowała się lepszymi właściwościami lub też nawet w pewnych warunkach szybszą zbieżnością, jak jest na przykład w przypadku jednej z zaproponowanych w pracy [11] równoległych implementacji algorytmu Gaussa-Seidela, który to algorytm dla niektórych zadań okazał się wyraźnie szybciej zbieżny niż jego podstawowa wersja sekwencyjna.

Jedną z bardzo interesujących propozycji równoległej implementacji obliczeń optymalizacyjnych jest metoda zaproponowana przez Chazana-Mirankera [1, 11]. Opiera się ona na równoległej optymalizacji kierunkowej prowadzonej jednocześnie przez $P = n$ procesorów (gdzie n – wymiar problemu optymalizacji) w jednym i tym samym kierunku, przy czym poszczególne procesory prowadzą tę optymalizację startując z pewnej, sukcesywnie modyfikowanej bazy n punktów określonych w przestrzeni R^n. W pracach poświęconych omawianej metodzie znajduje uzasadnienie stwierdzenie, iż jest ona zbieżna dla pewnej klasy ściśle wypukłych, podwójnie różniczkowalnych funkcji. Ponadto, jeżeli minimalizowana funkcja $f(\mathbf{x})$, gdzie $\mathbf{x} \in R^n$ jest funkcją kwadratową, wówczas metoda ta będzie zbieżna po wykonaniu maksymalnie n^2 optymalizacji kierunkowych, co przy założeniu, że n optymalizacji jest wykonywanych jednocześnie przez $P = n$ procesorów daje w efekcie zbieżność dla tego typu funkcji po n iteracjach, gdzie przez iterację rozumie się jeden pełny cykl algorytmu. Jak pokazały zaprezentowane w pracy [11] badania, metoda ta może prowadzić do bardzo interesujących rezultatów, w szczególności w odniesieniu np. do trudniejszych zadań optymalizacji (np. dla funkcji Rosenbrocka), dla których inne, prostsze metody zawodzą.

Innym, równie interesującym rozwiązaniem jest zaproponowany w pracy [11] algorytm równoległy oparty również na wykorzystaniu wiązki równoległej, lecz wykorzystujący dodatkowo pewną specyficzną właściwość tak prowadzonej optymalizacji, skutkującej mającą miejsce na każdym etapie algorytmu redukcją wymiaru przestrzeni poszukiwań, co dla pewnego typu problemów optymalizacji, gwarantuje uzyskanie zbieżności algorytmu do rozwiązania optymalnego w n krokach, lecz przy dwukrotnie mniejszym, w porównaniu do metody Chazana-Mirankera, nakładzie obliczeń.

Na szczególną uwagę zasługują również tak zwane algorytmy dwuetapowe. Powstały one również z myślą o wykorzystaniu do ich realizacji systemów równoległych, gdyż w naturalny sposób zakładają równoległe przeszukiwanie przestrzeni, w jakiej określone jest rozważane zadanie przy wykorzystaniu zbieżnej bądź równoległej wiązki kierunków [11, 13]. Są one połączeniem wykonywanych naprzemiennie algorytmów dwojakiego typu. Pierwszy z tych algorytmów, stanowiący tak zwaną iterację rozbieżną, ma za zadanie wygenerowanie rozproszonego zbioru rozwiązań aproksymujących rozwiązanie optymalne. W najprostszym przypadku może tu być wykorzystany algorytm realizujący poszukiwania w kierunkach wersorów kartezjańskiego układu współrzędnych (np. algorytm Gaussa-Seidela). Drugi z algorytmów polega z kolei na wytyczeniu w każdym z tych punktów kierunku poprawy, przeprowadzeniu optymalizacji w tych kierunkach oraz wyborze takiego spośród otrzymanych rozwiązań, które najlepiej aproksymuje rozwiązanie optymalne. Jeżeli kierunki te będą generowane niezależnie w każdym z punktów przy wykorzystaniu np. algorytmów newtonowskich, wówczas otrzyma się tzw. zbieżną wiązkę kierunków (iteracja zbieżna), jeżeli natomiast dla każdego z punktów generowany będzie jeden i ten sam wspólny kierunek to otrzymana wiązka będzie wiązką równoległą (iteracja równoległa) [11]. Jak z powyższego widać, w algorytmach tych tkwią duże potencjalne możliwości zwiększenia niezawodności obliczeń optymalizacyjnych w porównaniu do klasycznych algorytmów jednoetapowych. Wynika to głównie z tego, iż przeszukuje się dany obszar rozwiązania nie jednym, lecz wiązką kierunków, a ponadto przy generowaniu wiązki kierunków można posługiwać się nie jednym, lecz wieloma różnymi algorytmami, co może być ważne w przypadku trudnych zadań optymalizacji.

Przedmiotem niniejszego artykułu jest przedstawienie jeszcze innej możliwości równoległej realizacji obliczeń optymalizacyjnych zaproponowanej w pracy [11], będącej daleko idącą modyfikacją klasycznego algorytmu Gausa-Seidela. Modyfikacja ta polega na sukcesywnej aktualizacji przyjętej na początku obliczeń ortogonalnej bazy kierunków poszukiwań. Jak pokazano w pracy [11], podejście takie może prowadzić dla pewnych funkcji do bardzo dobrych wyników, konkurencyjnych w stosunku do rezultatów otrzymanych przy wykorzystaniu innych klasycznych metod optymalizacji podobnego typu.

2. Metoda Gaussa-Seidela

Algorytm Gaussa-Seidela jest jednym z najprostszych, a zarazem najbardziej naturalnym algorytmem rozwiązywania wielowymiarowych zadań programowania nieliniowego (ZPN) bez

ograniczeń, opartym na wykorzystaniu optymalizacji kierunkowej [3, 4, 7, 11, 12]. Cechą charakterystyczną tego algorytmu jest wykorzystanie stałej, niezmieniającej się w czasie realizacji obliczeń bazy kierunków określonej przez wersory kartezjańskiego układu współrzędnych. Kierunki te w przestrzeni R^n określone są następująco:

$$\xi_1 = [1, 0, 0, \ldots 0, 0]^{\mathrm{T}},$$

$$\xi_2 = [0, 1, 0, \ldots 0, 0]^{\mathrm{T}},$$

$$\ldots$$

$$\xi_n = [0, 0, 0, \ldots 0, 1]^{\mathrm{T}}.$$

Zadanie poszukiwania minimum funkcji bez ograniczeń można sformułować następująco:

$$f(\hat{\boldsymbol{x}}) = \min_{x \in R^n} f(\boldsymbol{x}), \tag{1}$$

gdzie: f: $R^n \rightarrow R$, przy czym zakłada się, że f jest funkcją ograniczoną od dołu.

Analizę efektywności równoległych realizacji omawianych algorytmów prowadzono w oparciu o przykładowe, mieszczące się w klasie problemów opisanych ogólną zależnością (1) zadanie programowania nieliniowego. Poniżej przedstawiono rozważane w artykule wersje algorytmu Gaussa-Seidela tak w implementacji sekwencyjnej, jak i analizowanych realizacjach równoległych. Do wyznaczania minimum funkcji w kierunku (w wersji sekwencyjnej oraz równoległej) wykorzystano algorytm Brenta [10] oparty na zastosowaniu metody interpolacji kwadratowej oraz metody złotego podziału. Jako kryterium stopu przy obliczaniu minimum w kierunku przyjęto spełnienie warunku:

$$\left|\lambda_{i+1} - \lambda_i\right| \leq \varepsilon_k\,, \tag{2}$$

gdzie: λ_i, λ_{i+1} oznaczają kolejno po sobie wyznaczane w algorytmie minimalizacji kierunkowej przesunięcia względem punktu $\boldsymbol{x}_0$ w kierunku $\xi(\boldsymbol{x} = \boldsymbol{x}_0 + \lambda\xi)$, gwarantujące zmniejszenie wartości funkcji $f(\boldsymbol{x}_0 + \lambda\xi)$. Natomiast generalnie dla każdej z analizowanych metod programowania nieliniowego kryterium stopu było spełnienie warunku:

$$\left\|\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\right\| \leq \varepsilon_m, \tag{3}$$

gdzie: $\boldsymbol{x}_{i+1}$, $\boldsymbol{x}_i$ oznaczają przybliżenia rozwiązania optymalnego wyznaczone w kolejnych krokach algorytmu, przy czym przez krok algorytmu rozumie się w przypadku metody Gaussa-Sedela wykonanie n zadań minimalizacji kierunkowych funkcji $f(\boldsymbol{x})$ we wszystkich kierunkach przyjętej bazy ortogonalnej, natomiast norma rozumiana jest w sensie normy euklidesowej, tzn.:

$$\|\boldsymbol{x}\| = \left(\sum_{i=1}^{n} (x_i)^2 \right)^{0,5}. \tag{4}$$

Dla wielkości określających odpowiednio dokładność realizacji optymalizacji kierunkowej (ε_k) oraz dokładność realizacji całej metody (ε_m) przyjęto następujące wartości

$$\varepsilon_k = 10^{-8},\ \varepsilon_m = 10^{-6}. \tag{5}$$

Jako punkt $\boldsymbol{x}_o$, z którego inicjowano obliczenia, przyjęto we wszystkich przeprowadzonych badaniach symulacyjnych punkt początkowy układu współrzędnych:

$$\boldsymbol{x}_0 = [0, 0, 0, \ldots 0, 0]^{\mathrm{T}}.$$

Dla rozważanego w artykule przykładu (15) badane algorytmy są praktycznie zbieżne do rozwiązania optymalnego niezależnie od wyboru punktu początkowego. Mogą jednak wystąpić pewne różnice w szybkości zbieżności (liczbie iteracji niezbędnych dla uzyskania wymaganej dokładności obliczeń), które nie mają jednak większego znaczenia z punktu widzenia rozważanej w artykule problematyki.

Algorytm sekwencyjny (AS)

Algorytm w implementacji sekwencyjnej sprowadza się do realizacji obliczeń według następujących kroków [11]:

1. dla $j = 1, 2, \ldots, n$ obliczyć kolejno $\hat{\lambda}_j$ minimalizujące:

$$\min_{\lambda_j}: f(\boldsymbol{x}_{j-1} + \lambda_j \xi_j), \tag{6}$$

oraz współrzędne nowego punktu:

$$\boldsymbol{x}_j = \boldsymbol{x}_{j-1} + \hat{\lambda}_j \xi_j, \tag{7}$$

gdzie: $f(\boldsymbol{x})$ – minimalizowana funkcja, $\boldsymbol{x} \in R^n$, $\xi_1, \xi_2, \ldots, \xi_n$ – baza wyjściowa utworzona z n wzajemnie ortogonalnych wersorów kartezjańskiego układu współrzędnych.

2. powyższe kroki powtarza się do momentu spełnienia kryterium stopu.

Przyjmując zadania minimalizacji kierunkowej za najmniejsze części algorytmu, które mogłyby być realizowane równolegle (jednocześnie), zadania tego nie da się w prosty sposób zrealizować równolegle, gdyż ma ono strukturę sekwencyjną. Dostosowanie omawianego algorytmu do równoległej architektury komputera (w wersji procesora wielordzeniowego, czy też klastra połączonych ze sobą komputerów) wymaga jego wcześniejszej modyfikacji. Zrównolegloną wersję omawianego algorytmu można sformułować w przedstawiony poniżej sposób [11].

Algorytm równoległy (AR1)

Algorytm w implementacji równoległej polega na realizacji obliczeń w następujących krokach [11, 12]:

1. każdy procesor oblicza $\hat{\lambda}_j$ minimalizujące funkcję $f(\boldsymbol{x})$ w jednym kierunku (gdy $n = P$, gdzie n – wymiar problemu, P – liczba procesorów) lub kolejno w kilku kierunkach (gdy $n > P$), $j = 1, 2, \ldots, n$:

$$\min_{\lambda_j}: f(\boldsymbol{x}_0 + \lambda_j \xi_j), \tag{8}$$

gdzie: $\boldsymbol{x}_0$ – wynik poprzedniej iteracji,

2. nowy punkt wyznacza się według zależności:

$$\boldsymbol{x} = \boldsymbol{x}_o + \sum_{j=1}^{n} \hat{\lambda}_j \xi_j \rightarrow \boldsymbol{x}_0, \tag{9}$$

3. powyższe kroki powtarza się do momentu spełnienia kryterium stopu.

Konsekwencją stosowania złożenia (9) może być wolniejsza zbieżność algorytmu, co zostało potwierdzone praktycznymi obliczeniami. Zaletą natomiast jest to, że tak sformułowany algorytm może być realizowany w sposób równoległy.

Przedstawiony powyżej algorytm równoległy, podobnie jak jego wyjściowa wersja sekwencyjna, zakłada korzystanie z niezmieniającej się w czasie bazy ortogonalnych kierunków poszukiwań. Zakładając możliwość jednoczesnej (równoległej) realizacji wielu zadań poszukiwania minimum optymalizowanej funkcji w wielu różnych kierunkach, można podjąć próbę sformułowania nowego algorytmu dopuszczającego modyfikację bazy kierunków. Idea ta legła u podstaw zaproponowanej w pracy

[11] nowej równoległej wersji analizowanego w artykule algorytmu. Algorytm w takiej wersji nie jest już *de facto* algorytmem Gaussa-Seidela. Można stwierdzić, iż zawiera on zarówno elementy algorytmu Gaussa-Seidela (optymalizacja względem n-wymiarowej bazy kierunków), jak i analizowanej w pracy [11] równoległej implementacji algorytmu Powella (dopuszczającej modyfikację wykorzystywanej bazy kierunków). Wersja sekwencyjna algorytmu Powella opisana jest w pracach [3, 4], natomiast zrównoleglony algorytm Gaussa-Seidela z modyfikowaną bazą kierunków został sformułowany poniżej.

Algorytm równoległy Gaussa-Seidela z modyfikacją bazy (AR2): Zaproponowany w pracy [11] algorytm równoległy z modyfikacją bazy sprowadza się do realizacji obliczeń według następujących kroków:

1. każdy procesor oblicza $\hat{\lambda}_j$ minimalizujące funkcję $f(\boldsymbol{x})$w jednym kierunku (gdy $n=P$, gdzie P – liczba procesorów) lub kolejno w kilku kierunkach (gdy $n>P$), $j=1,2,\ldots,n$:

$$\min_{\lambda_j}: f(\mathrm{x}_0+\lambda_j\xi_j), \tag{10}$$

gdzie: $\boldsymbol{x}_0$ – wynik poprzedniej iteracji,

2. nowy punkt wyznaczyć z:

$$\boldsymbol{x}=\boldsymbol{x}_0+\boldsymbol{e}_{\max}, \tag{11}$$

gdzie:

$$\boldsymbol{e}_{max_i}=\beta \max_j(|\boldsymbol{e}_i^j|),\quad \boldsymbol{e}^j=\hat{\lambda}_j\xi_j,\quad i,j=1,2,\ldots,n$$
$$\beta=1,\quad e_i^j\geq 0$$
$$\beta=-1,\ e_i^j<0$$

3. wyznaczyć współrzędne nowego kierunku:

$$\xi_j=\lambda/||\lambda||,\quad \text{gdzie } \lambda=\boldsymbol{x}-\boldsymbol{x}_0 \tag{12}$$

4. zmodyfikować bazę kierunków $\xi_1,\xi_2,\ldots,\xi_n$:

$$\xi_j=\xi_j^*+\alpha\xi,\quad i,j=1,2,\ldots,n \tag{13}$$

gdzie: $0<\alpha\ll 1$, ξ_j^* – przeorientowany znakowo kierunek ξ_j (za dodatni przyjmuje się zwrot zgodny ze zwrotem ξ),

5. powyższe kroki powtarza się do momentu spełnienia kryterium stopu.

Tak sformułowany algorytm znacząco różni się od równoległego algorytmu Gaussa-Seidela AR1. Następuje tutaj modyfikacja bazy kierunków w sposób nie zachowujący jej ortogonalności. Ponadto, ze względu na niezachowaną ortogonalność wykorzystywanej bazy kierunków, stosowanie prostego złożenia geometrycznego wyznaczanych równolegle przesunięć (uzyskanych dla poszczególnych kierunków bazowych) może prowadzić do „spowolnienia" algorytmu, jak też może, w krańcowych sytuacjach być przyczyną niezbieżności algorytmu. W celu uniknięcia tych niedogodności, zaproponowano złożenie tylko maksymalnych wartości wyznaczonych przesunięć, obliczanych względem poszczególnych składowych (wersorów) globalnego, kartezjańskiego układu współrzędnych (globalnego układu odniesienia). Generowany nowy kierunek wykorzystywany jest jedynie do modyfikowania istniejącej bazy według zależności (13), a nie tak jak to jest w metodzie Powella do tworzenia nowego kierunku bazowego. Wymagane przeorientowanie zwrotów kierunków ma zapewnić zawężanie się układu kierunków wokół wyznaczanych kolejno (w kolejnych krokach algorytmu) przybliżeń rozwiązania optymalnego. Początkowa wartość wykorzystywanego do modyfikacji bazy kierunków współczynnika α została empirycznie wyznaczona na 10^{-2}.

Na podstawie badań ustalono, że współczynnik α rzędu 10^{-2} jest odpowiedni tylko dla problemów o wymiarze nie większym niż 250. Dla problemów o większym wymiarze taka wartość współczynnika α była przyczyną zakłóceń w zbieżności algorytmu do rozwiązania optymalnego lub też skutkowała spowolnieniem szybkości zbieżności algorytmu. Problemy te znikały przy właściwym doborze wartości tego współczynnika dla danego wymiaru problemu n. Wartość współczynnika α dla problemów o większym wymiarze była dobierana na tym etapie badań eksperymentalnie. Generalnie przyczyną zaobserwowanych problemów w przebiegu algorytmu były problemy wynikające z niezachowania wzajemnej sprzężoności wektorów określających kierunki poszukiwań, wchodzących w skład sukcesywnie modyfikowanej bazy kierunków. Problemów tych uniknięto stosując sukcesywne odświeżanie bazy kierunków lub też wprowadzając na wzór metody Powella, dodatkowy warunek uzależniający modyfikację bazy od np. wartości wyznacznika macierzy kierunków.

Algorytm Gaussa-Seidela został zaimplementowany w dwóch środowiskach systemów równoległych: przy wykorzystaniu standardu OpenMP – w środowisku maszyn wielordzeniowych, oraz przy wykorzystaniu standardu MPI – na klastrze komputerów [6]. Obliczenia z biblioteką OpenMP zostały przeprowadzone na komputerze stacjonarnym z procesorem i5-3570k w systemie operacyjnym Ubuntu 12.04, natomiast obliczenia z protokołem MPI zostały przeprowadzone na klastrze uczelnianym składającym się z 16 identycznych maszyn (węzłów) z procesorem AMD Phenom II X6 1090T w systemie operacyjnym Fedora 17.

Poniżej przedstawiono wyniki numeryczne otrzymane na drodze praktycznej realizacji omówionych wyżej metod i algorytmów równoległych. Ocenę efektywności obliczeń równoległych prowadzono w oparciu o współczynnik przyspieszenia określony zależnością (14). Oznaczając mianowicie przez $t_{SEQ}(1,n)$ czas realizacji algorytmu sekwencyjnego (lub algorytmu równoległego realizowanego sekwencyjnie) na jednym wątku albo na jednym procesie, natomiast przez $t_{PAR}(P,n)$ czas równoległej realizacji algorytmu równoległego przy wykorzystaniu P wątków albo P procesów. Czasy te dotyczą rozwiązania tego samego n-wymiarowego problemu optymalizacji. Wartość współczynnika przyspieszenia obliczeń wyznaczano w oparciu o zależność:

$$S(P,n)=\frac{t_{SEQ}(1,n)}{t_{PAR}(P,n)}. \tag{14}$$

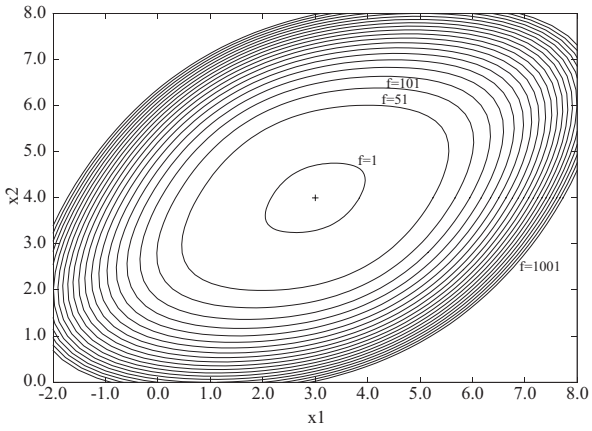
Obliczenia prezentowane poniżej przeprowadzone były na przykładzie następującego problemu optymalizacji:

$$\min_{x\in R^n} f(x)=\sum_{i=1}^{n}(2i-1)(x_i-(2+i))^4+\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(x_i-x_j+(j-i))^4, \tag{15}$$

gdzie n oznacza wymiar problemu optymalizacji.

Funkcja (15) jest typową funkcją testową, podobną do innych tego typu funkcji cytowanych w literaturze [13], skonstruowaną na potrzeby analizy porównawczej prowadzonej m.in. w pracy [11]. Jest ona funkcją ściśle wypukłą w całym obszarze określoności, spełniającą wszystkie podstawowe wymagania zawarte w warunkach stosowalności analizowanych metod optymalizacji. Na rys. 1 przedstawiono wykres poziomicowy funkcji (15) dla przypadku $n=2$.

Wyniki obliczeń przedstawione zostały na rysunkach: 2, 3 oraz 4. Prezentowane badania symulacyjne dotyczyły rozwiązania problemu (1) dla funkcji $f(x)$ określonej zależnością (15), przy $n=500$. Rysunki 2 oraz 3 dotyczą algorytmu AR1, przy czym rys. 2 obrazuje zależność wartości współczynnika przyspieszenia obliczeń w funkcji liczby wątków (OpenMP), natomiast rys. 3. przedstawia zależność wartości tego współczynnika od liczby procesów (MPI). W tym przypadku obliczenia zrealizowane były na 16 węzłowym (96 rdzeniowym) klastrze, przy czym wyszcze-



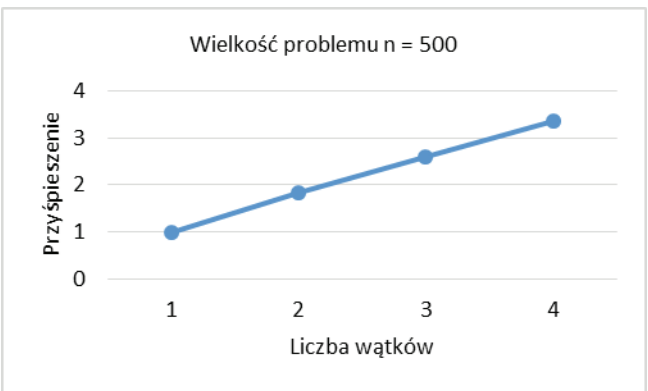
Rys. 1. Funkcja testowa: $n = 2$
Fig. 1. Benchmark function: $n = 2$

gólnione w obliczeniach procesy realizowane były na różnych rdzeniach wchodzących w skład węzłów klastra. Rys. 4 dotyczy natomiast algorytmu AR2 i przedstawia, podobnie jak rys. 2, zależność wartości współczynnika przyspieszenia obliczeń w funkcji liczby wątków. Jak wynika z rysunków 2 oraz 4, algorytmy AR1 oraz AR2 niewiele się różnią jeżeli chodzi o uzyskane wartości przyspieszenia obliczeń, co jest raczej zrozumiałe z uwagi na przyjęty w artykule, często stosowany w praktyce, sposób obliczania wartości współczynnika przyspieszenia. Za punkt odniesienia przy wyznaczaniu jego wartości, czyli jako czas sekwencyjnej realizacji algorytmu, przyjęto czas realizacji zrównoleglonej wersji algorytmu na jednym wątku/procesie.

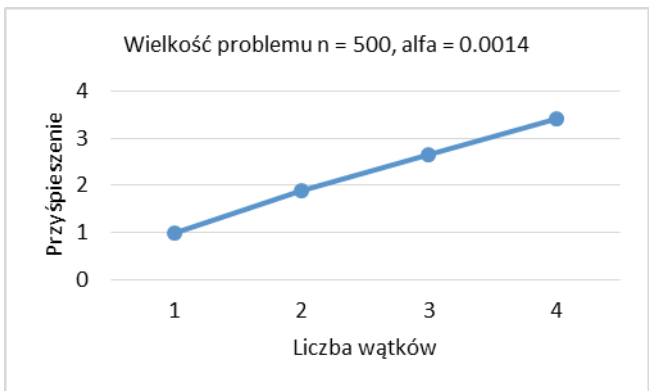
Różnice pomiędzy analizowanymi wersjami algorytmu Gaussa-Seidela będą możliwe do zaobserwowania jedynie wówczas, gdy do ich porównania przyjmie się wskaźnik odzwierciedlający rzeczywistą czasochłonność obliczeń. Dobrą bezwzględną miarą może być tutaj liczba iteracji niezbędnych do przeprowadzenia w celu uzyskania takiej samej dokładności rozwiązania (dla takich samych parametrów zadania). Jest to tym bardziej uzasadnione, iż czasochłonność pojedynczych iteracji dla wszystkich analizowanych algorytmów dla takiej samej liczby wątków/procesów powinna być zbliżona. Nakład obliczeń mierzony liczbą iteracji jaką otrzymano w przypadku algorytmów AS, AR1 oraz AR2 (dla $n = 500$) przedstawiono na rys. 5. Rysunek ten obrazuje bardzo interesującą zaletę analizowanego algorytmu z modyfikacją bazy (AR2), a mianowicie dla rozważanych przykładów okazał się on szybciej zbieżny zarówno w stosunku do podstawowej wersji równoległej tego algorytmu (AR1), jak też, co jest niezmiernie istotne, w stosunku do jego oryginalnej wersji sekwencyjnej (AS). Wynik ten zaobserwowany w pracy [11] dla niewielkich wymiarów analizowanych przykładowych problemów optymalizacji potwierdził się również dla zadań o znacznie większym wymiarze.

3. Poprawiona metoda Gaussa-Seidela z modyfikacją bazy

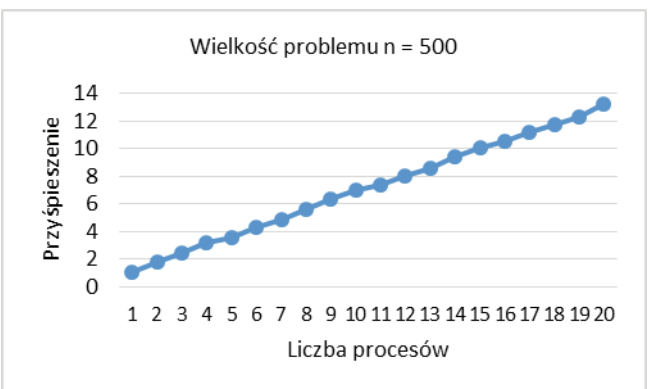
Algorytm Gaussa-Seidela z modyfikacją bazy, dla odpowiednio dobranego α, pozwalał na otrzymanie rozwiązania przy wyraźnie mniejszej liczbie iteracji niż algorytm sekwencyjny oraz podstawowy równoległy algorytm Gaussa-Seidela. Eks-



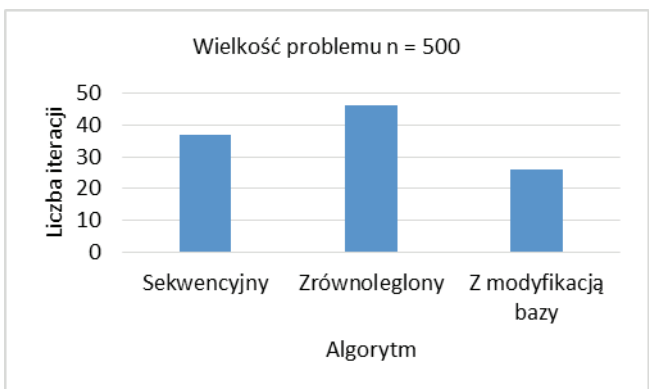
Rys. 2. Równoległy algorytm Gaussa-Seidela (AR1): n = 500 (OpenMP)
Fig. 2. Parallel Gauss-Seidel algorithm (AR1): $n = 500$. (OpenMP)



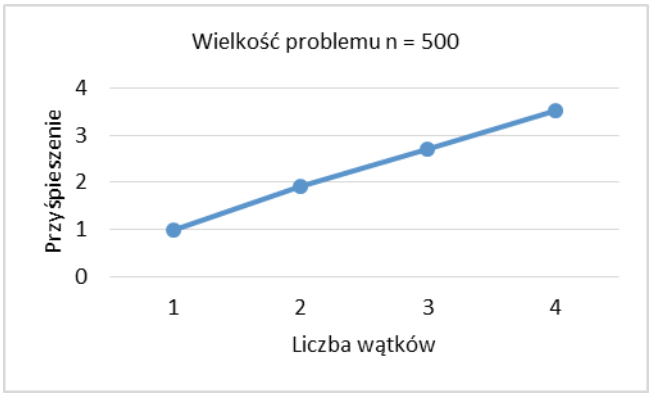
Rys. 4. Równoległy algorytm Gaussa-Seidela z modyfikacją bazy kierunków (AR2): $n = 500$ (OpenMP)
Fig. 4. Parallel Gauss-Seidel algorithm with modification of search directions (AR2): $n = 500$ (OpenMP)



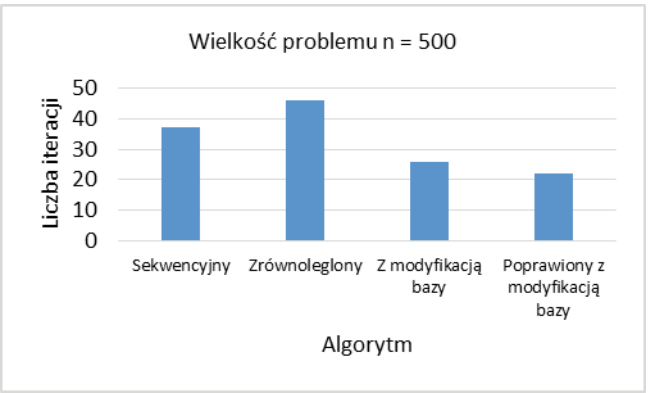
Rys. 3. Równoległy algorytm Gaussa-Seidela (AR1): $n = 500$, $P = 1,2,\ldots,20$ (klaster, MPI)
Fig. 3. Parallel Gauss-Seidel algorithm (AR1): $n = 500$, $P = 1,2,\ldots,20$ (cluster, MPI)



Rys. 5. Algorytmy Gaussa-Seidela (AS, AR1, AR2): szybkość zbieżności (OpenMP)
Fig. 5. Parallel Gauss-Seidel algorithms (AS, AR1, AR2): speed of convergence (OpenMP)



Rys. 6. Równoległy algorytm Gaussa-Seidela (AR2m): *n* = 500 (OpenMP)
Fig. 6. Parallel Gauss-Seidel algorithm (AR2m): *n* = 500. (OpenMP)



Rys. 7. Algorytmy Gaussa-Seidela (AS, AR1, AR2, AR2m): szybkość zbieżności (OpenMP)
Fig. 7. Parallel Gauss-Seidel algorithms (AS, AR1, AR2, AR2m): speed of convergence (OpenMP)

perymentalny dobór wartości współczynnika α dla różnych wielkości problem jest jednak działaniem przybliżonym i nie zawsze prowadził do zadowalających rezultatów w odniesieniu do uzyskiwanych praktycznie szybkości zbieżności algorytmu. W zakresie doboru wartości tego współczynnika wymagane jest wobec tego prowadzenie dalszych badań teoretycznych, które być może pozwolą w przyszłości określać wartość tego współczynnika w sposób optymalny.

Inną wprowadzoną dodatkowo modyfikacją algorytmu było przywracanie bazy kierunków do stanu pierwotnego po wykonaniu pewnej, założonej z góry liczby iteracji (odświeżanie bazy). Dzięki tej modyfikacji zauważona została poprawa właściwości algorytmu – zawsze znajdował on rozwiązanie optymalne, lecz kosztem pewnego spowolnienia obliczeń – zwiększenia liczby iteracji.

Poprawiony algorytm równoległy Gaussa-Seidela z modyfikacją bazy (AR2m)

1. każdy procesor oblicza $\hat{\lambda}_j$ minimalizujące funkcję $f(\boldsymbol{x})$ w jednym kierunku (gdy $n=P$, gdzie P – liczba procesorów) lub kolejno w kilku kierunkach (gdy $n>P$), $j = 1, 2, ..., n$:

$$\min_{\lambda_j}: f(\boldsymbol{x}_0+\lambda_j\xi_j), \tag{16}$$

gdzie: $\boldsymbol{x}_0$ – wynik poprzedniej iteracji,

2. nowy punkt wyznaczyć z:

$$\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{e}_{\max}, \tag{17}$$

gdzie:

$$\boldsymbol{e}_{max_i} = \beta \max_j (|\boldsymbol{e}_i^j|), \quad \boldsymbol{e}^j = \hat{\lambda}_j\xi_j, \quad i,j = 1,2,...,n$$
$$\beta = 1, \quad \boldsymbol{e}_i^j \geq 0$$
$$\beta = -1, \; \boldsymbol{e}_i^j < 0$$

3. sprawdzić warunek

$$l \; mod 10 = 0 \; oraz \; l > 0 \tag{18}$$

gdzie: l – numer iteracji, 10 – przyjęta częstotliwość odświeżania bazy.
Jeżeli warunek jest spełniony to wykonaj krok (4),
jeżeli warunek nie jest spełniony to przejdź do kroku (5).

4. odśwież bazę kierunków

$$\xi = \xi_{pocz} \tag{19}$$

gdzie: ξ_{pocz} – początkowa baza kierunków,
przejdź do kroku (9)

5. sprawdzić warunek

$$det \; \Xi \geq 0.45 \tag{20}$$

gdzie $\Xi = [\xi_1, \xi_2, ... \xi_n]$ – macierz kierunków.
Jeżeli warunek jest spełniony to przejdź do kroku (6),
jeżeli warunek nie jest spełniony to przejdź do kroku (9).

6. wyznaczyć współrzędne nowego kierunku:

$$\xi_j = \lambda / ||\lambda||, \quad \text{gdzie } \lambda = \boldsymbol{x} - \boldsymbol{x}_0 \tag{21}$$

7. zmodyfikować bazę kierunków:

$$\xi_j = \xi_j^* + \alpha\xi, \quad i,j = 1,2,...,n \tag{22}$$

gdzie:
$0 < \alpha \ll 1$, ξ_j^* – przeorientowany znakowo kierunek ξ_j
(za dodatnim przyjmuje się zwrot zgodny ze zwrotem ξ)

8. znormalizować bazę kierunków:

$$\xi_j = \xi_j / ||\xi_j|| \tag{23}$$

9. powyższe kroki powtarza się do momentu spełnienia kryterium stopu.

Obliczenia dla zmodyfikowanego algorytmu Gaussa-Seidela z modyfikacją bazy zostały przeprowadzone przy wykorzystaniu standardu OpenMP. Wartość współczynnika α wynosiła 10^{-2}, odświeżanie bazy było wykonywane co 10 iteracji. Badania przeprowadzono dla wielkości problemu $n = 500$. Najlepsze wyniki uzyskano dla przyjętej w warunku (20) wartości 0,45.

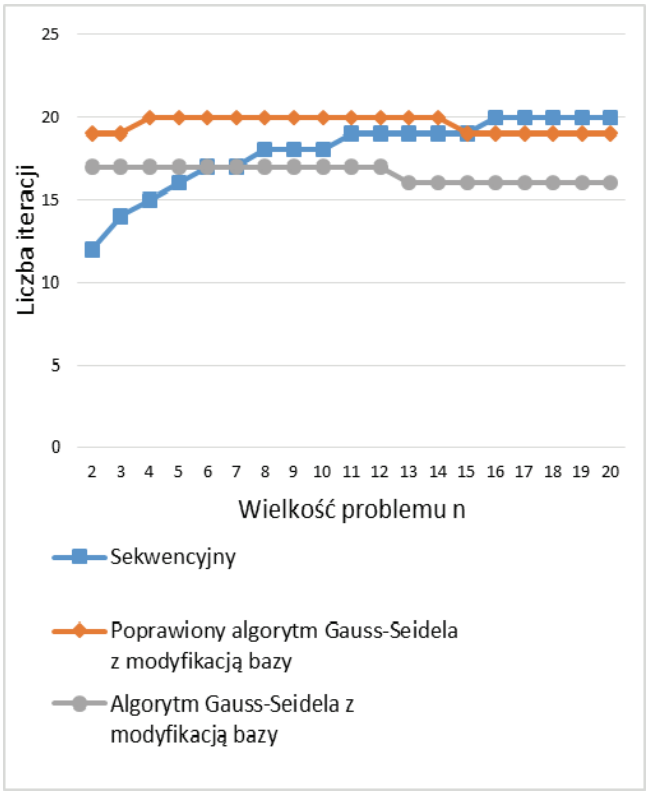
Wyniki otrzymane dla tej wersji metody przedstawiono na rysunkach 6 oraz 7, przy czym rys. 6 obrazuje zależność wartości współczynnika przyspieszenia obliczeń w funkcji liczby wątków, natomiast rys. 7 przedstawia liczbę iteracji otrzymaną dla algorytmu AR2m na tle analogicznych rezultatów otrzymanych dla wszystkich wcześniej omawianych wersji metody Gaussa-Seidela. Rys. 6 pokazuje, iż algorytm AR2m poddaje się zrównolegleniu równie dobrze jak pozostałe algorytmy, natomiast rys. 7 wskazuje iż jest on pod względem czasochłonności obliczeń najbardziej efektywnym algorytmem spośród wszystkich wersji rozważanych w artkule.

Na rysunkach 8 oraz 9 przedstawiono z kolei przebieg zależności liczby iteracji niezbędnych dla osiągnięcia założonej dokładności obliczeń dla analizowanego przykładu obliczeniowego odpowiednio dla problemów o wymiarze $n = 2, 3, 4, ..., 20$ (rys. 8) oraz problemów o wymiarze $n = 50, 100, ..., 950$ (rys. 9). Rysunki te pokazują, że dla niewielkich wymiarowości analizowanego problemu optymalizacji proponowany algorytm wykazuje przewagę w porównaniu do sekwencyjnej jak i podstawowej równoległej jego wersji, natomiast dla dużych wymiarowości problemu przewagę tą można zachować wprowadzając okresowe odświeżanie bazy kierunków.

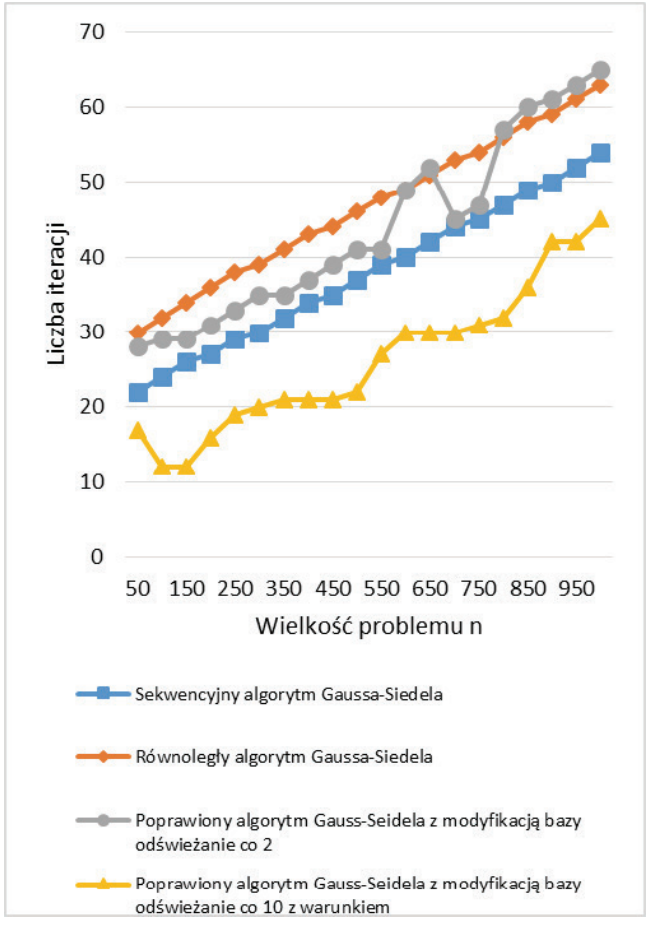
4. Podsumowanie i dalsze kierunki badań

Z przeprowadzonych i częściowo zaprezentowanych w niniejszym artykule badań wynika, że dla zadań o znacznie większym wymiarze w porównaniu do rozważań prowadzonych w pracy [11] (gdzie analizowano zadania o wymiarze przeważnie do $n = 20$), to znaczy np. dla $n = 500$ i więcej dało się również w wielu analizowanych przypadkach zaobserwować wyraźną, pod względem szybkości zbieżności, przewagę algorytmu Gaussa-Seidela z modyfikacją bazy zarówno w stosunku do podstawowej wersji równoległej tego algorytmu, jak też, co jest niezmiernie istotne, w stosunku do jego oryginalnej wersji sekwencyjnej. Dalsze badania obejmą w przyszłości jeszcze bardziej złożoną pod względem wymiarowości grupę problemów optymalizacji, jak też tzw. trudniejsze zadania optymalizacji stanowiące pewne wyzwanie również dla innych znanych efektywnych algorytmów i metod optymalizacji.

Innym problemem jest zagadnienie analizy i określenia teoretycznych warunków zbieżności rozważanego w artykule algorytmu bardzo mocno uzależnionych od sposobu generowania kierunków poszukiwań. Wiadomo, że jeżeli jakiekolwiek dwa kierunki spośród kierunków tworzących macierz kierunków Ξ, gdzie $\Xi = [\xi_1, \xi_2, ..., \xi_n]$, będą względem siebie liniowo zależne, to wówczas wyznacznik tej macierzy będzie równy zero: $\det \Xi = 0$. Liniowa zależność jakichkolwiek dwóch, lub większej liczby kierunków zawartych w modyfikowanej na bieżąco bazie kierunków może skutkować dużymi trudnościami w osiągnięciu zbieżności algorytmu w ogóle, a w szczególności trudnościami w osiągnięciu zbieżności algorytmu do rozwiązania optymalnego. Stąd zagwarantowanie spełnienia warunku: $\det \Xi > 0$ przy każdorazowej modyfikacji macierzy kierunków dokonywanej w trakcie działania algorytmu może mieć bardzo duże znaczenia. W szczególności, warunek ten może przyjąć ostrzejszą formę, a mianowicie: $\det \Xi \geq \beta$, gzie β jest przyjętą graniczną dopuszczalną wartością jaką może przyjąć wyznacznik macierzy Ξ. W dotychczasowych badaniach problem określenia wartości współczynnika β analizowany był jedynie na gruncie eksperymentalnym, i pokazał że ma on niebagatelny wpływ na szybkość zbieżności algorytmu. W dalszych badaniach podjęte będą działania zmierzające do teoretycznego określenia wartości tego współczynnika lub też do określenia bardziej sformalizowanego warunku gwarantującego zbieżność analizowanego w artykule algorytmu równoległego z modyfikacją bazy np. na wzór warunków sformułowanych dla metody Powella [3, 4].



Rys. 8. Algorytmy Gaussa-Seidela: $n = 2, 3, ..., 20$ (OpenMP)
Fig. 8. Gauss-Seidel algorithms: $n = 2, 3, ..., 20$ (OpenMP)



Rys. 9. Algorytmy Gaussa-Seidela: $n = 50, 100, ..., 1000$ (OpenMP)
Fig. 9. Gauss-Seidel algorithms: $n = 50, 100, ..., 1000$ (OpenMP)

Bibliografia

1. Chazan D., Miranker W.L., *A Nongradient and Parallel Algorithm for Unconstrained Minimization*, "SIAM Journal on Control", vol.8, No. 2, May 1970, 207–217. DOI: 10.1137/0308015.
2. Dennis J.E., Torczon V., *Direct Search Methods on Parallel Machines*, SIAM Journal on Optimization, Vol. 1, No.4, November 1991, 448–474. DOI: 10.1007/978-3-642-48417-9_2.

3. Findeisen Wł., Szymanowski J., Wierzbicki A., *Metody obliczeniowe optymalizacji*, Wydawnictwo Politechniki Warszawskiej, 1973.
4. Findeisen Wł., Szymanowski J., Wierzbicki A., *Teoria i metody obliczeniowe optymalizacji*. PWN, Warszawa 1980.
5. Kaliczyńska M., Sadecki J., *Obliczenia równoległe – klastry obliczeniowe*, Zeszyty Naukowe PO, s. Elektryka, z. 57, 2006, 101–114.
6. Karbowski A., Niewiadomska-Szynkiewicz E., *Programowanie równoległe i rozproszone*, Oficyna Wydawnicza PW, Warszawa 2009.
7. Korytkowski A., Ziółko M., *Metody optymalizacji z ćwiczeniami laboratoryjnymi*, Wydawnictwo AGH, Kraków 1992.
8. Lewis A., *Parallel Optimization Algorithms for Continuous, Non-linear Numerical Simulation*, PhD thesis, 2004, Griffith University, Brisbane, Australia.
9. Molga M., Smutnicki Cz., *Test functions for optimization needs*, 2005, www.robertmarks.org/Classes/ENGR5358/Papers/functions.pdf.
10. Press W. H., Flannery B. P., Teukolsky S.A., Vetterling W. T., *Numerical Recipes, The Art of Scientific Computing*, Cambridge University Press, 1986.
11. Sadecki J., *Algorytmy równoległe optymalizacji i badanie ich efektywności; systemy równoległe z rozproszoną pamięcią*, Oficyna Wydawnicza Politechniki Opolskiej, Opole 2001.
12. Sadecki J., *Równoległe implementacje algorytmu Gaussa-Seidela w środowisku OpenMP*, PAK, 2011, nr 03, 301–304.
13. Sobczyk J., Wierzbicki A.P., *Obliczenia równoległe w optymalizacji liniowej i nieliniowej. Algorytm pulsacyjny w optymalizacji nieliniowej*, [w:] I Konferencja Granty-Automatyka'95. Warszawa, 27–29.06.1995.

Efficiency Analysis of Some Parallel Implementations of the Gauss-Seidel Algorithm

Abstract: The paper presents the results of the efficiency analysis of some parallel implementations of Gauss-Seidel algorithm. The main idea of the presented method consists in successive modification of the search directions used in the computations. This modification is performed on the basis of solutions of local optimization subproblems received for all stages of the algorithm. The analyzed algorithm enable to achieve a good efficiency of parallel computation in terms of speed of convergence and value of speedup factor in comparison to standard sequential and parallel implementation of Gauss-Seidel method. Parallel computation were implemented in the multicore processor and multiprocessor cluster.

Keywords: optimizatiom algorithms, parallel computation, parallel optimization algorithm

mgr inż. Marek Machaczek

marek.ma89@gmail.com

Dyplom magistra inżyniera w zakresie informatyki uzyskał w 2013 r. na Wydziale Automatyki, Elektrotechnik i Informatyki Politechniki Opolskiej. Obecnie jest doktorantem Politechniki Opolskiej w dyscyplinie Automatyka i Robotyka. Głównym kierunkiem jego badań są obliczenia równoległe.

dr hab. inż. Jan Sadecki, prof. PO

j.sadecki@po.opole.pl

Jest długoletnim pracownikiem Politechniki Opolskiej. Stopień doktora nauk technicznych (1988 r.) oraz doktora habilitowanego (2004 r.) uzyskał w dyscyplinie Automatyka i Robotyka na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. Zajmuje się przetwarzaniem równoległym i rozproszonym, systemami równoległymi, klastrami obliczeniowymi.