Jan SADECKI
POLITECHNIKA OPOLSKA, WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI I INFORMATYKI
ul. Sosnkowskiego 31, 45-272 Opole

Równoległe implementacje algorytmu Gaussa-Seidela w środowisku OpenMP

Dr hab. inż. Jan SADECKI

Studia ukończył w 1978 r. Doktorat obronił w 1988 r. na Wydziale Elektroniki Politechniki Warszawskiej, natomiast stopień doktora habilitowanego uzyskał na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej w 2004 r. Od 2005 r. pracuje na stanowisku prof. nadzw. w Politechnice Opolskiej. Autor ponad 50 prac naukowych. Specjalizuje się w dziedzinie obliczeń równoległych i rozproszonych w zastosowaniu do rozwiązywania złożonych zadań i problemów optymalizacji.

e-mail: j.sadecki@po.opole.pl

Streszczenie

W artykule przedstawiono przykładowe rezultaty analizy efektywności równoległych realizacji algorytmu Gaussa-Seidela zaimplementowanych w środowisku procesorów wielordzeniowych. Jak pokazano, standardowa równoległa implementacja tego algorytmu, prowadzi do gorszych w sensie szybkości zbieżności wyników w porównaniu do sekwencyjnej wersji tej metody. Zaproponowana nowa wersja równoległa metody Gaussa-Seidela posiada analogiczną szybkość zbieżności jak jej realizacja sekwencyjna, zachowując przy ty łatwość implementacji równoległej. W artykule przedstawiono przykładowe rezultaty obliczeń przeprowadzonych przy wykorzystaniu procesora czterordzeniowego. Rozważana implementacja algorytmu Gaussa-Seidela posiada też możliwości jej zastosowania dla szerszej niż rozważana w pracy klasy problemów optymalizacji.

Słowa kluczowe: klastry, algorytmy równoległe, procesory wielordzeniowe.

Parallel implementation of Gauss-Seidel algorithm by using OpenMP

Abstract

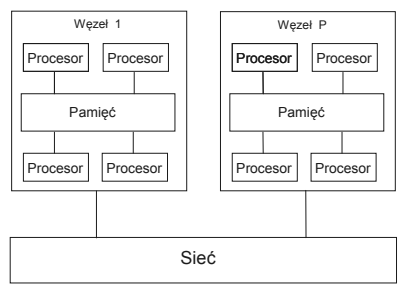
The paper presents results of the efficiency analysis for some parallel realization of optimisation algorithms in multicore processors. The results concern a simple Gauss-Seidel optimization algorithm. In the paper both standard parallel and new parallel implementations of the Gauss-Seidel algorithm are presented. As it is pointed out, the standard parallel algorithm leads to worse numerical results (in terms of the rate of computation convergence) than the sequential version of this algorithm. The new parallel algorithm achieves the same numerical efficiency of computations as the sequential algorithm and, additionally, can be aesily implemented in multicore processors. It is prooved that, for the quadratic optimization problem, the modified parallel Gauss-Seidel algorithm leads to the same computational results as for the sequential implementation of the method. Some examples of parallel implementations of the method in fourcore processors are presented. The proposed new algorithm enables achieving good efficiency of parallel computations both in terms of the execution time and the speedup factor value. The new algorithm can also be used to solve broader classes of optimization problems, which in the nearest neighbourhood of the optimal solution can be sufficiently precisely approximated by the square function.

Keywords: clusters, parallel algorithms, multicore processors.

1. Wstęp

W ostatnich latach obserwuje się wyraźną tendencje odchodzenia od wysoko specjalizowanych architektur superkomputerów w stronę technologii klastrowych, co daje niejednokrotnie znaczne obniżenie kosztów sprzętu komputerowego z zachowaniem podobnej lub nawet osiągnięciem wyższej mocy obliczeniowej [2, 7]. Według pochodzącej z czerwca 2010 roku edycji listy Top500 [www.top500.org] 424 superkomputery spośród 500 umieszczonych na tej liście najszybszych maszyn na świecie to komputery o architekturze klastrowej, co stanowi ok. 84.8 % ich ogólnej liczby. Współczesne klastry posiadają najczęściej strukturę hierarchiczną (generalnie dwupoziomową). Składają się one z połączonych szybką siecią wielu jedno lub wieloprocesorowych węzłów, przy czym w roli węzłów występują procesory wielordzeniowe lub też, co jest aktualnie nowym, lecz bardzo obiecującym rozwiązaniem specjalizowane procesory GPGPU (rys. 1). Globalna liczba rdzeni w przypadku najszybszych tego typu klastrów dochodzi nawet do kilkuset tysięcy. Taka architektura klastra wymaga zastosowania złożonych, niejednorodnych narzędzi wspomagających programowanie tego typu maszyn. Na poziomie węzłów wykorzystuje się najczęściej interfejs MPI, natomiast w obrębie węzłów, w przypadku procesorów o architekturze wielordzeniowej wymagane jest tworzenia aplikacji wielowątkowych. Wygodnym narzędziem wspomagającym tworzenie tego typu aplikacji może być np. środowisko OpenMP [2].

Przy programowaniu złożonych aplikacji numerycznych korzysta się bardzo często z szeroko dostępnych gotowych, efektywnych i dobrze opracowanych bibliotek procedur i funkcji realizujących różnego typu metody i algorytmy numeryczne. Wiele z tych bibliotek jest jednak nadal dostosowanych do realizacji jedynie w środowisku procesorów jednordzeniowych. Biorąc pod uwagę opisane wyżej trendy i tendencje w zakresie rozwoju komputerów dużej mocy, koniecznym staje się opracowanie nowych wersji tych bibliotek, dostosowanych do automatycznej realizacji również w środowiskach procesorów wielordzeniowych. Podejmując badania w tym zakresie przeanalizowano szereg wybranych algorytmów optymalizacji. Dla algorytmów tych opracowano ich implementacje numeryczne dostosowane do wielordzeniowej architektury procesorów. W artykule zamieszczono wybrane wyniki przeprowadzonych badań oraz analiz numerycznych. Analizując szereg różnych implementacji tych algorytmów poszukiwano takich rozwiązań uniwersalnych, które gwarantowałyby osiągnięcie możliwie największej efektywności obliczeniowej (najlepszego przyśpieszenia obliczeń). Tak opracowane moduły programowe stanowić mogą zalążek przyszłej uniwersalnej biblioteki optymalizacyjnej dostosowanej do najpopularniejszych aktualnie rozwiązań w zakresie architektury superkomputerów.



Rys. 1. Architektura klastra
Fig. 1. Cluster architecture

Efektywność tworzonych aplikacji równoległych zależy w dużym stopniu zarówno od rodzaju rozwiązywanego problemu, użytej metody jak też od architektury systemu równoległego. Istnieją algorytmy dla których konstruowanie implementacji równoległych może być zadaniem relatywnie prostym, jak np. niektóre problemy algebry liniowej, są też jednak takie problemy, dla których zrównoleglenie obliczeń może prowadzić do algorytmów o gorszych w stosunku do wersji sekwencyjnej własnościach numerycznych, (jak np. analizowany w artykule prosty algorytm Gaussa-Seidela stosowany do rozwiązywania zadań programowania nieliniowego) chociaż znane są też algorytmy których zrówno-

leglenie prowadzi do poprawy ich własności (jak np. algorytm Neldera-Meada [6]). W artykule rozważano problem możliwości poprawy efektywności równoległych realizacji dla algorytmów, których równoległe implementacje charakteryzują się gorszą efektywnością w porównaniu do ich odpowiedników sekwencyjnych. Rozważania zmierzające do poprawy efektywności równoległych implementacji tego typu algorytmów prowadzono na przykładzie wspomnianego algorytmu Gaussa-Seidela w zastosowaniu do rozwiązywania zadań kwadratowy, chociaż implementacje te mogą być też rozszerzone na zadania szerszego typu, które można dobrze aproksymować przy pomocy formy kwadratowej. Zaproponowane w artykule zależności pozwalają wyznaczyć na bazie kolejnych rozwiązań uzyskanych na drodze równoległej realizacji rozważanego algorytmu wartości rozwiązań identyczne z tymi, jakie gwarantuje sekwencyjna (szybciej zbieżna) jego realizacja. Zależności te dla metody Gaussa-Seidea mogą być – w zastosowaniu do zadań kwadratowych - dosyć prosto wyprowadzone. Podobne podejście można jednak zastosować również w odniesieniu do innych, algorytmów i metod optymalizacji bazujących na poszukiwaniu ekstremum funkcji kolejno - w implementacjach sekwencyjnych, lub jednocześnie – w implementacjach równoległych względem przyjętej w tych metodach bazy kierunków (stałej lub zmiennej). Jako przykłady tych metod można tutaj wymienić np. metody Rosenbrocka, Powella, Zangwilla itp. W przypadku tego typu metod zależności wiążące rozwiązania dla równoległych oraz sekwencyjnych implementacji będą jednak znacznie bardziej złożone.

W Instytucie Automatyki i Informatyki PO prowadzone są szersze badania dotyczące metodologii oraz praktyki tworzenia efektywnych aplikacji równoległych dla złożonych zadań i problemów praktycznych. Przykładowo dotyczą one zagadnień modelowania i prognozowania procesu rozprzestrzeniania się i kumulacji zanieczyszczeń w atmosferze [4] czy też zastosowania klastrów komputerowych w systemach wizyjnych robotów mobilnych [5].

2. Metoda Gaussa-Seidela

Algorytm Gaussa-Seidela jest w zasadzie jednym z najprostszych i najbardziej naturalnych algorytmów poszukiwania ekstremum funkcji wielowymiarowych, opartym na wykorzystaniu optymalizacji kierunkowych, realizowanych kolejno względem kierunków określonych przez wektory przyjętej bazy ortogonalnej. Jako bazę kierunków przyjmuje się najczęściej wersory kartezjańskiego układu współrzędnych.

Rozważania prezentowane w artykule prowadzone są na bazie typowego problemu poszukiwania minimum funkcji bez ograniczeń, który można sformułować następująco:

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

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

Algorytm Gaussa-Seidela w standardowej wersji sekwencyjnej można ogólnie sformułować w sposób następujący [1, 6]:

Algorytm sekwencyjny:

(i) dla j=1,2,...,n obliczyć kolejno λ_i minimalizujące:

$$\min_{\lambda_j}: \quad f(\mathbf{x}_{j-1} + \lambda_j \boldsymbol{\xi}_j),$$

oraz współrzędne nowego punktu:

$$\mathbf{x}_j = \mathbf{x}_{j-1} + \hat{\lambda}_j \boldsymbol{\xi}_j,$$

gdzie: $f(\mathbf{x})$ - funkcja celu, $\mathbf{x}\in R^n$, ξ_1, ξ_2,...,ξ_n - baza wyjściowa utworzona z n wzajemnie ortogonalnych wektorów,

(ii) powtarzać krok (i) do momentu spełnienia kryterium stopu.

Przyjmując za najmniejszą "porcję" algorytmu, która mogłaby być realizowana równolegle zadanie minimalizacji kierunkowej (równoległość gruboziarnista), algorytmu opisanego powyżej nie można bezpośrednio zrealizować w systemie wieloprocesorowym. Wprowadzając naturalną jego modyfikację polegającą na prowadzeniu niezależnych, startujących z tego samego punktu minimalizacji kierunkowych, realizowanych względem kierunków takiej samej bazy ortogonalnej, równoległą wersję omawianego algorytmu można sformułować w sposób następujący:

Algorytm równoległy:

(i) każdy procesor oblicza λ_i minimalizujące funkcję $f(\mathbf{x})$ w jednym kierunku (gdy n=P, gdzie P - liczba procesorów) lub kolejno w kilku kierunkach (gdy n>P), j=1,2,...,n:

$$\min_{\lambda_j}: \quad f(\mathbf{x}_0 + \lambda_j \boldsymbol{\xi}_j),$$

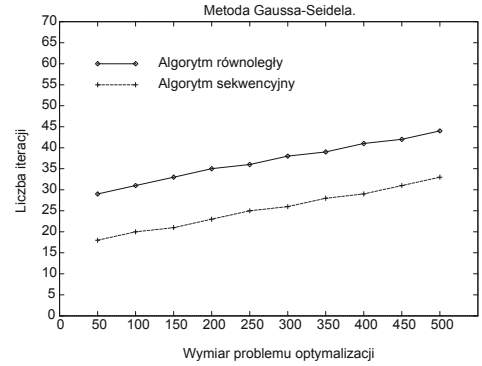
gdzie $\mathbf{x}_0$ - wynik z poprzedniej iteracji,

(ii) na bazie otrzymanych w punkcie (i) wartości przesunięć kierunkowych wyznacza się współrzędne nowego punktu:

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

(iii) powtarzać kroki (i) oraz (ii) do momentu spełnienia kryterium stopu.

Newralgicznym etapem tego algorytmu jest zadanie opisane w punkcie (ii), które można sformułować w sposób następujący: jak wyznaczyć współrzędne nowego przybliżenia dla poszukiwanego rozwiązania na podstawie znajomości określonych równolegle optymalnych przesunięć λ_i, j=1,2,...,n, otrzymanych względem kierunków przyjętej bazy ortogonalnej? Informacja jaką daje algorytm równoległy ma charakter rozproszony ("horyzontalny"). Aby określić na jej podstawie współrzędne nowego punktu można posłużyć się wykorzystaniem pewnych procedur interpolacyjno-ekstrapolacyjnych. Przykładem może tu być zastosowany w powyższym algorytmie prosty schemat ekstrapolacyjny, określający współrzędne nowego przybliżenia na drodze geometrycznego złożenia przesunięć lokalnych. Konsekwencją stosowania tego typu procedur może być, potwierdzona praktycznymi obliczeniami, wyraźnie wolniejsza zbieżność algorytmu równoległego w porównaniu do jego wersji sekwencyjnej. Problem ten zobrazowano na rys. 2, na którym przedstawiono niezbędną dla uzyskania zamierzonej dokładności obliczeń liczbę iteracji w funkcji wymiaru problemu optymalizacji n, dla obu wersji algorytmu.



Rys. 2. Liczba iteracji dla równoległej oraz sekwencyjnej realizacji algorytmu Gaussa-Seidela

Fig. 2. Number of iterations for parallel and sequential realization of Gauss-Seidel algorithms

Obliczenia wykonano na bazie następującego przykładowego problemu minimalizacji funkcji wielu zmiennych:

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

Analiza teoretyczna różnic jakie występują pomiędzy sekwencyjnym algorytmem Gaussa-Seidela a jego wersją równoległą przeprowadzona zostanie na przykładzie pewnej zawężonej klasy funkcji kwadratowych. Zakłada się mianowicie, że funkcja $f(\mathbf{x})$, gdzie $\mathbf{x}\in R^n$ jest ograniczoną od dołu funkcją wypukłą klasy C^2 oraz że w bliskim otoczeniu minimum można ją aproksymować formą kwadratową o postaci:

$$f(\mathbf{x}) = a + \mathbf{b}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}, \tag{3}$$

przy czym macierz drugich pochodnych **A** jest macierzą symetryczną, dodatnio określoną.

Oznaczając przez $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2,...,\boldsymbol{\xi}_n$ bazę wyjściową utworzoną z wzajemnie ortogonalnych wektorów, pokrywających się w analizowanym przypadku z wersorami kartezjańskiego układu współrzędnych można określić optymalne przesunięcie $\hat{\lambda}_i$ otrzymane w wyniku minimalizacji funkcji $f(\mathbf{x})$ realizowanej względem jednego z kierunków $\boldsymbol{\xi}_i$, i=1,2,...,n, przy arbitralnie wybranym punkcie startowym $\mathbf{x}_0$. Współrzędne otrzymanego w wyniku tej minimalizacji punktu określone będą zależnością:

$$\mathbf{x} = \mathbf{x}_0 + \hat{\lambda}_i\boldsymbol{\xi}_i. \tag{4}$$

Podstawiając (4) do wyrażenia (3), otrzymamy:

$$f(\mathbf{x}) = a + \mathbf{b}^{\mathrm{T}}\mathbf{x}_0 + \frac{1}{2}\mathbf{x}_0^{\mathrm{T}}\mathbf{A}\mathbf{x}_0 + \hat{\lambda}_i\mathbf{b}^{\mathrm{T}}\boldsymbol{\xi}_i + \frac{1}{2}\hat{\lambda}_i\mathbf{x}_0^{\mathrm{T}}\mathbf{A}\boldsymbol{\xi}_i + \\ + \frac{1}{2}\hat{\lambda}_i\boldsymbol{\xi}_i^{\mathrm{T}}\mathbf{A}\mathbf{x}_0 + \frac{1}{2}\hat{\lambda}_i^2\boldsymbol{\xi}_i^{\mathrm{T}}\mathbf{A}\boldsymbol{\xi}_i. \tag{5}$$

Różniczkując obustronnie wyrażenie (5) względem $\hat{\lambda}_i$ oraz korzystając z założenia o symetryczności macierzy **A**, otrzymamy:

$$\frac{\partial f(\mathbf{x})}{\partial \hat{\lambda}_i} = b_i + \mathbf{a}_i^{\mathrm{T}}\mathbf{x}_0 + \hat{\lambda}_i a_{ii}, \tag{6}$$

gdzie: b_i - i-ta składowa wektora **b**, $\mathbf{a}_i$ - i-ta kolumna macierzy **A**, a_{ii} - element diagonalny macierzy **A**.

Przyrównując pochodną (6) do zera, otrzymamy zależność określającą optymalne przesunięcie otrzymane w wyniku minimalizacji funkcji $f(\mathbf{x})$ prowadzonej względem kierunku $\boldsymbol{\xi}_i$:

$$\hat{\lambda}_i = \frac{-b_i - \mathbf{a}_i^{\mathrm{T}}\mathbf{x}_0}{a_{ii}}. \tag{7}$$

Wykorzystując (7) można sformułować ogólną zależność określającą optymalne przesunięcie otrzymane w wyniku minimalizacji zrealizowanej względem wszystkich n kierunków przyjętej bazy ortogonalnej dla obydwu algorytmów, tzn. algorytmu sekwencyjnego oraz algorytmu równoległego.

Algorytm sekwencyjny

W algorytmie sekwencyjnym minimalizacja realizowana jest kolejno względem wszystkich kierunków przyjętej bazy, rozpoczynając np. od kierunku $\boldsymbol{\xi}_1$, przy czym minimalizacja względem kierunku $\boldsymbol{\xi}_i$ rozpoczyna się w punkcie $\mathbf{x}_0^{i-1}$, otrzymanym w wyniku minimalizacji realizowanej w kierunku $\boldsymbol{\xi}_{i-1}$. Przesunięcia $\hat{\lambda}_i$ określone będą zależnością [6]:

$$\hat{\boldsymbol{\lambda}} = \mathbf{D}^{-1}(-\mathbf{b} - \mathbf{A}\mathbf{x}_0 - \mathbf{L}\hat{\boldsymbol{\lambda}}), \tag{8}$$

gdzie: **A** = **L**+**D**+**U**, **L** - dolna macierz trójkątna, **D** - macierz diagonalna, **U** - górna macierz trójkątna (**U**=$\mathbf{L}^{\mathrm{T}}$).

Wyznaczając z zależności (8) przesunięcie $\hat{\boldsymbol{\lambda}}$, oraz wprowadzając oznaczenie $\boldsymbol{\lambda}_{seq} = \hat{\boldsymbol{\lambda}}$, otrzymuje się:

$$\boldsymbol{\lambda}_{seq} = (\mathbf{L} + \mathbf{D})^{-1}(-\mathbf{b} - \mathbf{A}\mathbf{x}_0). \tag{9}$$

Zależność ta przedstawia przesunięcie otrzymane przy zastosowaniu do minimalizacji funkcji opisanej wyrażeniem (3) jednego pełnego cyklu sekwencyjnego algorytmu Gaussa-Seidela, przy założeniu że minimalizacja prowadzona jest kolejno względem przyjętej bazy n ortogonalnych kierunków $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2,...,\boldsymbol{\xi}_n$, pokrywających się z układem współrzędnych kartezjańskich. Jak łatwo zauważyć, przy innej kolejności prowadzenia minimalizacji kierunkowych, zależność (9) przyjmie inną postać. Wyniki te byłyby identyczne jedynie w przypadku spełnienia warunku: **A**=**D**.

Algorytm równoległy

W algorytmie równoległym minimalizacja funkcji (3) realizowana jest poprzez jednoczesne poszukiwanie minimum tej funkcji względem wszystkich kierunków przyjętej bazy ortogonalnej, przy tym samym punkcie początkowym $\mathbf{x}_0$. Ponieważ powyższe zadania minimalizacji kierunkowej realizowane są niezależnie od siebie, stąd też zależność określającą przesunięcie otrzymane w wyniku przeprowadzenia powyższych n optymalizacji kierunkowych można otrzymać poprzez proste uogólnienie wrażenia (7). Wprowadzając oznaczenie $\boldsymbol{\lambda}_{par} = \hat{\boldsymbol{\lambda}}$, otrzymuje się:

$$\boldsymbol{\lambda}_{par} = \mathbf{D}^{-1}(-\mathbf{b} - \mathbf{A}\mathbf{x}_0). \tag{10}$$

Wykorzystując zależności (9) oraz (10) można określić proste związki jakie zachodzą pomiędzy przesunięciami otrzymywanymi w wyniku przeprowadzenia jednego cyklu (n) optymalizacji kierunkowych realizowanych według algorytmu sekwencyjnego $\boldsymbol{\lambda}_{seq}$ oraz algorytmu równoległego $\boldsymbol{\lambda}_{par}$, przy założeniu, że minimalizacja ta dotyczy klasy funkcji opisanych zależnością (3):

$$\boldsymbol{\lambda}_{par} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{D})\boldsymbol{\lambda}_{seq}, \tag{11}$$

$$\boldsymbol{\lambda}_{seq} = (\mathbf{L} + \mathbf{D})^{-1}\mathbf{D}\boldsymbol{\lambda}_{par}. \tag{12}$$

Zależności te mogą być wykorzystane przy projektowaniu obliczeń równoległych, szczególnie np. w przypadku gdy algorytm równoległy charakteryzuje się wolniejszą zbieżnością od jego wersji sekwencyjnej, jak to było np. pokazane powyżej na bazie analizowanego przykładu numerycznego. W takim przypadku, bazując na wyznaczonych równolegle wartościach $\boldsymbol{\lambda}_{par}$, można, korzystając z zależności (12), obliczyć wartości przesunięć $\boldsymbol{\lambda}_{seq}$ gwarantujące uzyskanie identycznej (pod względem liczby iteracji) zbieżności jak w algorytmie sekwencyjnym. Ponieważ macierz **L**+**D** jest macierzą trójkątną, stąd też algorytm jej odwracania tzn. wyznaczania macierzy $(\mathbf{L}+\mathbf{D})^{-1}$ nie jest numerycznie zadaniem zbyt skomplikowanym i może być docelowo również realizowany równolegle. Ponadto dla analizowanej klasy funkcji macierz (**L**+**D**) jest macierzą stałą, stąd też zadanie jej odwracania może być zrealizowane tylko jednokrotnie np. na początku całego procesu minimalizacji.

3. Obliczenia w środowisku OpenMP

Korzyści jakie może dać wykorzystanie w obliczeniach zależności (12) zobrazowano na przykładzie zastosowania równoległego algorytmu Gaussa-Seidela do rozwiązania zadania opisanego zależnością (2), mieszczącego się w klasie funkcji określonych zależnością (3).

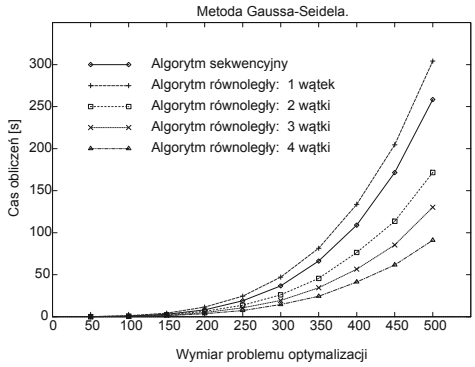
Wyniki obliczeń zrealizowanych przy wykorzystaniu środowiska OpenMP [2] przedstawiono na rysunkach 3-6. Rysunki te dotyczą zależności odpowiednio czasu obliczeń oraz wartości współczynnika przyspieszenia obliczeń od wymiaru analizowanego problemu optymalizacji. Wartości współczynnika przyspieszenia obliczeń odniesiono do czasu realizacji algorytmu ściśle se-

kwencyjnego. Wszystkie obliczenia wykonano przy wykorzystaniu procesora czterordzeniowego.

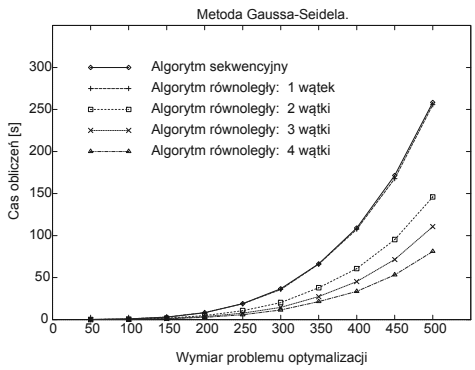
Wartość współczynnika przyspieszenia obliczeń $s(n,p)$ była wyznaczana w oparciu o zależność:

$$s(n,p) = \frac{t_{seq}(n)}{t_{par}(n,p)}, \tag{13}$$

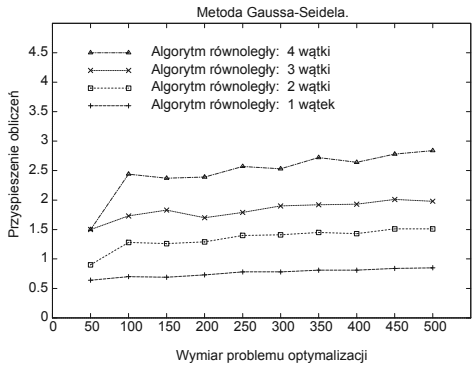
gdzie n oznacza wielkość problemu optymalizacji, natomiast przez p oznaczono liczbę wykorzystanych do obliczeń rdzeni procesora 4-rdzeniowego; t_{seq} oraz t_{par} oznaczają natomiast odpowiednio czas realizacji algorytmu Gaussa-Seidela w wersji sekwencyjnej oraz równoległej.



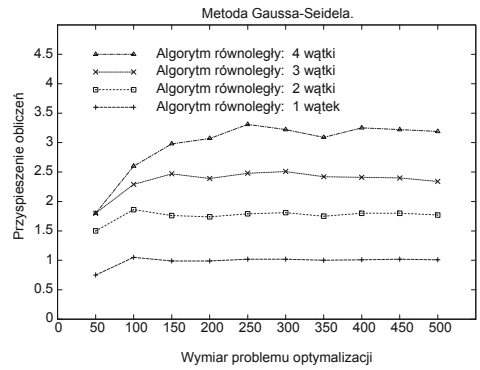
Rys. 3. Czas obliczeń dla standardowego równoległego oraz sekwencyjnego algorytm Gaussa-Seidela
Fig. 3. Computing time for standard parallel and sequential Gauss-Seidel algorithms



Rys. 4. Czas obliczeń dla zmodyfikowanego równoległego oraz sekwencyjnego algorytm Gaussa-Seidela
Fig. 4. Computing time for modified parallel and sequential Gauss-Seidel algorithms



Rys. 5. Przyspieszenie obliczeń dla standardowego równoległego oraz sekwencyjnego algorytm Gaussa-Seidela
Fig. 5. Speedup of computation for standard parallel and sequential Gauss-Seidel algorithms



Rys. 6. Przyspieszenie obliczeń dla zmodyfikowanego równoległego oraz sekwencyjnego algorytm Gaussa-Seidela
Fig. 6. Speedup of computation for modified parallel and sequential Gauss-Seidel algorithms

Rysunki 3 oraz 5 dotyczą opisanego wyżej klasycznego algorytmu równoległego, natomiast rysunki 4 oraz 6 dotyczą algorytmu równoległego wspartego modyfikacją przesunięć realizowaną w oparciu o zależność (12). Korzyści wynikające z wprowadzonej modyfikacji są bardzo wyraźnie widoczne. Tak powstały algorytm charakteryzuje się analogiczną szybkością zbieżności jak jego wersja sekwencyjna, co znacznie zmniejsza czas obliczeń, podnosząc jednocześnie efektywność jego implementacji równoległej.

4. Wnioski

Prezentowany artykuł dotyczy zagadnień związanych z badaniem możliwości poprawy efektywności równoległych realizacji dla pewnej grupy algorytmów optymalizacji, których równoległe implementacje charakteryzują się gorszą efektywnością w porównaniu do ich odpowiedników sekwencyjnych. Na przykładzie algorytmu Gaussa-Seidela pokazano, iż zaproponowany sposób modyfikacji jego realizacji równoległej może powadzić do znaczącej poprawy jego efektywności. Prezentowane w artykule rozważania dotyczyły co prawda stosunkowo wąskiej klasy problemów optymalizacji, jednak jak pokazują prowadzone badania istnieją realne możliwości rozszerzenia zaproponowanego rozwiązania na szerszą grupę problemów, które można w pobliżu minimum dostatecznie dokładnie aproksymować formą kwadratową.

5. Literatura

[1] Findeisen W., Szmanowski J., Wierzbicki A.: Teoria i metody obliczeniowe optymalizacji, PWN, 1980.
[2] Karbowski A., Niewiadomska-Szynkiewicz E. (eds): Programowanie równoległe i rozproszone. Oficyna Wydawnicza Politechniki Warszawskiej, 2009.
[3] Kumar V., Grama A., Gupta A., Karypis G.: Introduction to Parallel Computing, Addison Wesley, 2002.
[4] Majer M.: Process of Modeling Pollutants Dispersion in the Atmosphere. Education, Research, Innovation – ERIN 2009, Conference, Ostrava 2009.
[5] Podpora M., Sadecki J.: Biologically reasoned point-of-interest image compression for mobile robots, ISBN: 978-3-642-03201-1, DOI: 10.1007/978-3-642-03202-8_29, pp 389-401, Springer, 2009.
[6] Sadecki J.: Algorytmy równoległe optymalizacji i badanie ich efektywności; systemy równoległe z rozproszoną pamięcią, Studia i monografie, Politechnika Opolska, Opole 2001.
[7] Wyrzykowski R.: Klastry komputerów PC i architektury wielordzeniowe: budowa i wykorzystanie. AOW EXIT, Warszawa, 2009.

otrzymano / received: 26.11.2010
przyjęto do druku / accepted: 02.02.2011 *artykuł recenzowany*