ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996

Seria: INFORMATYKA z. 31 Nr kol. 1348

Hafed ZGHIDI
Wojciech BURLIKOWSKI

RÓWNOLEGŁE WYZNACZENIE WARTOŚCI WŁASNYCH MACIERZY Z WYKORZYSTANIEM MECHANIZMÓW ZDALNEGO WYWOŁANIA PROCEDUR

Streszczenie. Niniejszy artykuł przedstawia przykład równoległego wyznaczenia wartości własnych macierzy o dużych rozmiarach z wykorzystaniem mechanizmów zdalnego wywołania procedur (RPC) w sieci komputerowej w środowisku systemu Unix.

PARALLEL EIGENSYSTEMS SOLVING WITH REMOTE PROCEDURE CALL MECHANISMS

Summary. The paper presents an example of parallel eigenvalues determination of large matrices using the remote procedure call (RPC) mechanisms in Unix network.

LE CALCUL DES VALEURS PROPRES DE GRANDS SYSTÈMES PARALLÈLEMENT UTILISANT LES MÉCANISMES D'EXÉCUTION À LOIN (RPC)

Resumé. L'article présente l'example d'utulisation des mécanismes d'exécution des procédures à loin (RPC) dans les résaux locaux sous système Unix, pour le calcul parallèl des valeurs propres de grand systèmes.

1. Wstęp

Analizę stabilności systemów opisanych liniowymi układami równań różniczkowo-algebraicznych, przyjmujących postać równania stanu oraz wyjścia:

$$\frac{d}{dt}x(t)=A\,x(t)+B\,u(t)$$
$$y(t)=C\,x(t)+D\,u(t) \tag{1}$$

gdzie: $u(t)$ - wektor wejściowy [r×1],
$x(t)$ - wektor zmiennych stanu [n×1],
$y(t)$ - wektor wyjściowy [s×1],

przeprowadza się najczęściej w oparciu o wyznaczenie rozkładu wartości własnych macierzy stanu układu A [n×n] na płaszczyźnie zespolonej. Są one rozwiązaniami tzw. równania charakterystycznego macierzy kwadratowej A stopnia n o postaci:

$$\det(\lambda 1-A)=0 \tag{2}$$

gdzie: λ - wartość własna,
1 - macierz jednostkowa stopnia n.

W praktycznych algorytmach wyznaczania wartości własnych bardzo często wykorzystuje się własność podobieństwa macierzy, powodującą, iż macierz

$$A'=PAP^{-1} \tag{3}$$

ma takie same wartości własne jak macierz A, gdy macierz P jest nieosobliwa. Wynika to z przekształconego równania charakterystycznego dla macierzy A':

$$\det(\lambda' 1-A')=0$$
$$\Downarrow$$
$$\det[P\,(\lambda' 1-A)\,P^{-1}]=0 \tag{4}$$
$$\Downarrow$$
$$\det(P)\det(\lambda' 1-A)\det(P^{-1})=0$$

które jest spełnione, jeśli $\det(\lambda' 1-A)=0$, gdyż $\det(P)\neq 0, \det(P^{-1})\neq 0$, ponieważ macierz P jest nieosobliwa. Dowodzi to równości wartości własnych macierzy A, A' : $\lambda=\lambda'$, gdyż spełniają one identyczne równanie charakterystyczne.

Konieczność przeprowadzenia powyższej analizy występuje m.in. w przypadku badania stabilności dużych systemów elektroenergetycznych. Ich zlinearyzowane wokół ustalonego punktu pracy równania przyjmują postać równań stanu i wyjścia z zależności (1), przy czym rozmiar macierzy A często przekracza n=1000. Z tego względu w trakcie analizy najczęściej ograniczamy się do wyznaczania tzw. elektromechanicznych wartości własnych, związanych z ruchem wirników generatorów synchronicznych zainstalowanych w systemie, gdyż mają one decydujący wpływ na jego zapas stabilności. Służą do tego specjalne algorytmy iteracyjne (AESOPS, PEALS), które korzystają z ich wstępnych przybliżeń wyznaczonych na podstawie ogólnych algorytmów wyznaczania wartości własnych (Lanczos, QR) dla przypadków zredukowanych.

Wykorzystanie ogólnych algorytmów dla pełnego układu równań stanu systemu jest ograniczone dużymi rozmiarami jego macierzy oraz czasem trwania obliczeń. Próbie pokonania tych ograniczeń w przypadku algorytmu QR poprzez wykorzystanie obliczeń równoległych jest poświęcony poniższy artykuł.

2. Operacje pomocnicze

Implementacja algorytmu QR wymaga uwzględnienia następujących własności macierzy stanu systemu elektroenergetycznego:

a) niesymetrii wartości jej elementów ($a_{ij} \neq a_{ji}$),

b) złego jej uwarunkowania numerycznego, co w powiązaniu z niesymetrią elementów powoduje dużą wrażliwość otrzymanych wartości własnych na małe zmiany jej elementów oraz błędy zaokrągleń,

c) dużych rozmiarów (n>1000).

Z tego względu konieczne jest przeprowadzenie dwóch wstępnych operacji, wykorzystujących przekształcenia macierzy stanu przez podobieństwo, którymi są:

- balansowanie - służące do zbliżenia norm odpowiadających sobie wierszy i kolumn macierzy, co pozwala zmniejszyć błąd numerycznego wyznaczania wartości własnych.

  W celu uniknięcia wprowadzania dodatkowych błędów elementy macierzy transformacyjnej muszą być całkowitymi potęgami podstawy systemu obliczeniowego (radix base) na zastosowanym komputerze (najczęściej 2, dla dużych systemów IBM (mainframe) 16),

- redukcja - do macierzy górnej prawietrójkątnej (tzw.macierzy Hessenberga) poprzez wykorzystanie metody eliminacji Gaussa, co umożliwia znaczne ograniczenie liczby operacji arytmetycznych w procesie wyznaczania wartości własnych (z rzędu $O(n^3)$ do $O(n^2)$).

3. Przekształcenie QR

Algorytm wyznaczania wartości własnych macierzy A wykorzystujący przekształcenie QR polega na jej iteracyjnym przekształceniu do postaci $A=QR$, gdzie macierz:

Q - jest macierzą ortogonalną, górną prawietrójkątną,

R - jest macierzą górną trójkątną,

$$\begin{vmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n-1} & a_{2,n} \\ 0 & a_{3,2} & a_{3,3} & \cdots & a_{3,n-1} & a_{3,n} \\ 0 & 0 & a_{4,3} & \cdots & a_{4,n-1} & a_{4,n} \\ . & . & . & . & . & . \\ 0 & 0 & 0 & \cdots & a_{n,n-1} & a_{n,n} \end{vmatrix} = \begin{vmatrix} q_{1,1} & q_{1,2} & q_{1,3} & \cdots & q_{1,n-1} & q_{1,n} \\ q_{2,1} & q_{2,2} & q_{2,3} & \cdots & q_{2,n-1} & q_{2,n} \\ 0 & q_{3,2} & q_{3,3} & \cdots & q_{3,n-1} & q_{3,n} \\ 0 & 0 & q_{4,3} & \cdots & q_{4,n-1} & q_{4,n} \\ . & . & . & . & . & . \\ 0 & 0 & 0 & \cdots & q_{n,n-1} & q_{n,n} \end{vmatrix} \begin{vmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \cdots & r_{1,n-1} & r_{1,n} \\ 0 & r_{2,2} & r_{2,3} & \cdots & r_{2,n-1} & r_{2,n} \\ 0 & 0 & r_{3,3} & \cdots & r_{3,n-1} & r_{3,n} \\ 0 & 0 & 0 & \cdots & r_{4,n-1} & r_{4,n} \\ . & . & . & . & . & . \\ 0 & 0 & 0 & \cdots & 0 & r_{n,n} \end{vmatrix} \quad (5)$$

i dokonywaniu tej samej operacji na kolejnych macierzach otrzymanych z wymnażania wyznaczonych składników w odwrotnej kolejności. Prowadzi to do ciągu macierzowego

$$\begin{aligned} A &= A_1 = Q_1 R_1 \\ A_i &= Q_i R_i = R_{i-1} Q_{i-1} \quad i=2,3,\ldots \end{aligned} \quad (6)$$

Niezmienność wartości własnych wynika z podobieństwa kolejnych macierzy ciągu. W granicy ($i \to \infty$) ciąg $\{A_i\}$ dąży do macierzy górnej trójkątnej z wartościami własnymi położonymi na diagonali oraz, w przypadku wartości własnych zespolonych sprzężonych, będących wartościami własnymi bloków [2×2] leżących na jej diagonali.

W procesie dekompozycji macierzy A_i na kolejne macierze Q_i,R_i korzystamy z transformacji ortogonalnej *opisanej macierzą S*, wykorzystującej tzw.obroty Givensa, które pozwalają efektywnie wykorzystać własność prawietrójkątności macierzy A_i, zachowywaną w trakcie procesu opisanego zależnością (6). W skład pojedynczej iteracji procesu dekompozycji wchodzą następujące, powtarzane k-krotnie, etapy:

a) wyznaczenie $\cos(\upsilon_k)$, $\sin(\upsilon_k)$ zgodnie z zależnością:

dla k-tego kroku (k=1(1)n-1), które pozwolą zdefiniować macierz obrotu Givensa

$$r=\sqrt{a_{k,k}^2+a_{k+1,k}^2}$$

$$\cos(\vartheta_k)=\frac{a_{k,k}}{r} \tag{7}$$

$$\sin(\vartheta_k)=\frac{a_{k+1,k}}{r}$$

$$G^{(k)}=\begin{array}{c} \begin{array}{cccccccc} (1) & (2) & \dots & (k) & (k+1) & \dots & (n-1) & (n) \end{array} \\ \begin{bmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 & 0 \\ . & . & . & . & . & . & . & . \\ 0 & 0 & \dots & \cos(\vartheta_k) & -\sin(\vartheta_k) & \dots & 0 & 0 \\ 0 & 0 & \dots & \sin(\vartheta_k) & \cos(\vartheta_k) & \dots & 0 & 0 \\ . & . & . & . & . & . & . & . \\ 0 & 0 & \dots & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \end{array} \begin{array}{c} (1) \\ (2) \\ . \\ (k) \\ (k+1) \\ . \\ (n-1) \\ (n) \end{array} \tag{8}$$

b) wymnożenia aktualnej postaci macierzy $A_i^{(k)}$ przez macierz obrotu Givensa $G^{(k)}$, w celu wyzerowania elementu $a_{k+1,k}$, co powoduje zmianę jedynie k,k+1 wierszy tej macierzy, opisaną zależnościami:

$$\begin{aligned} a_k^{(k+1)} &= \cos(\vartheta_k)a_k^{(k)}+\sin(\vartheta_k)a_{k+1}^{(k)} \\ a_{k+1}^{(k+1)} &= -\sin(\vartheta_k)a_k^{(k)}+\cos(\vartheta_k)a_{k+1}^{(k)} \end{aligned} \tag{9}$$

gdzie: (k),(k+1) - oznacza krok algorytmu,

a_k - oznacza k-ty wiersz macierzy $A_i^{(...)}$.

Uwzględnienie prawietrójkątności kolejnych macierzy A_i powoduje, iż zależność (9) musi być realizowana jedynie na kolumnach o numerach (j≥k),

c) aktualizacja macierzy transformacji $S^{(k)}$ poprzez jej wymnożenie przez macierz obrotu Givensa $G^{(k)}$ wg zależności identycznej do (6). Jej budowa pozwala na dokonanie tej operacji jedynie na kolumnach o numerach (j≤k+1). Macierz ta przed rozpoczęciem procesu jest macierzą jednostkową stopnia n, natomiast po jego zakończeniu staje się macierzą dolną prawietrójkątną.

Po wypełnieniu (n-1) kroków powyższego procesu otrzymujemy macierze:

$$A^{(n-1)}=SA_i=R_i$$
$$S^{(n-1)}=S=Q_i^{-1} \tag{10}$$
$$S=\sum_{k=1}^{n-1} G^{(k)}$$

Dla otrzymania macierzy do kolejnej, (i+1)-szej, iteracji algorytmu QR, zgodnie z zależnością (6), należy wyznaczyć macierz $A_{i+1}=R_iQ_i$.

Korzystamy w tym celu z własności macierzy ortogonalnej, jaką jest macierz $S=Q_i^{-1}$, wyrażonej zależnością:

$$Q_i=(S)^{-1}=(S)^T=\begin{bmatrix} s_{1,1} & s_{1,2} & 0 & 0 & . & 0 \\ s_{2,1} & s_{2,2} & s_{2,3} & 0 & . & 0 \\ s_{3,1} & s_{3,2} & s_{3,3} & s_{3,4} & . & 0 \\ . & . & . & . & . & . \\ s_{n-1,1} & s_{n-1,2} & s_{n-1,3} & s_{n-1,4} & . & s_{n-1,n} \\ s_{n,1} & s_{n,2} & s_{n,3} & s_{n,4} & . & s_{n,n} \end{bmatrix}^T \tag{11}$$

gdzie: T - oznacza transpozycję macierzy,

Prowadzi to do zależności:

$$A_{i+1}=R_i(Q_i^{-1})^T=R_i(S)^T \tag{12}$$

W praktyce nie dokonujemy transpozycji, zaś mnożenie macierzy w zależności (9) przeprowadzamy wg wzoru:

$$a_{j,m}=\sum_{t=1}^{n} r_{j,t}s_{m,t} \tag{13}$$

Uwzględnienie budowy biorących w tym mnożeniu udział macierzy:

- R - macierz górna trójkątna,
- S - macierz dolna prawietrójkątna,

pozwala ogólny wzór na mnożenie macierzy (13) zapisać w postaci (14) w celu uniknięcia działań na elementach zerowych.

$$a_{j,m} = \sum_{\substack{t=j \\ t \le n-1}}^{m+1} r_{j,t} s_{m,t} \tag{14}$$

Po inspekcji wartości elementów poddiagonalnych tej macierzy pod kątem wyznaczenia wartości własnych przechodzimy do kolejnej iteracji algorytmu QR-zal. (6).

4. Równoległy algorytm przekształcenia QR

Równoległa realizacja powyższego algorytmu przy wykorzystaniu (P+1) komputerów, na które składa się P jednostek wykonawczych oraz jedna sterująca, wymaga:

- podziału macierzy A wg kolumn w jednostce centralnej i przesłania tak uporządkowanych elementów na komputery wykonawcze (p-ty komputer posiada elementy p-tej, p+P-tej, p+2P-tej ...etc. kolumny),
- utworzenia na każdym z komputerów wykonawczych odpowiadającej mu części macierzy S,

Wykonanie tych działań, na wstępie każdej iteracji algorytmu QR opisanej zależnością (6), pozwala zrealizować proces dekompozycji macierzy A w następujących etapach:

a) na k-tym kroku procesu na komputerze, który posiada elementy k-tej kolumny, wyznaczane są współczynniki obrotu Givensa zgodnie z zależnością (7) i przesyłane do jednostki centralnej, która rozprowadza je do pozostałych komputerów,

b) otrzymane wartości są wykorzystywane do przeprowadzania operacji opisanej zależnością (9) na każdym z komputerów w stosunku do elementów k,k+1 wiersza, które się na nim znajdują.

Po wykonaniu (n-1) kroków powyższego algorytmu przeprowadzamy na każdej z maszyn wymnażanie znajdujących się na niej części macierzy R,S:

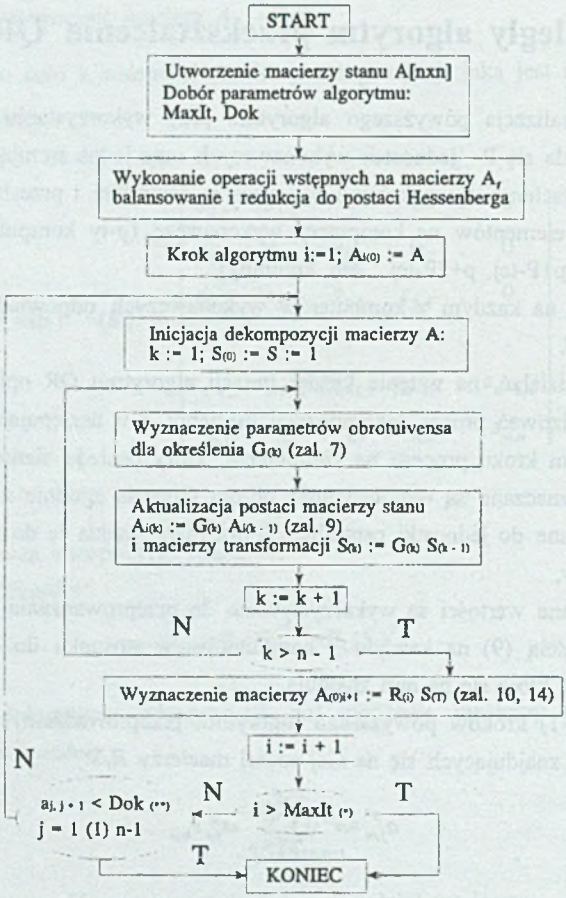
$$a_{j,m}^{p} = \sum_{t=p,p+P,p+2P\ldots}^{t(p)} r_{j,t} s_{m,t} \tag{15}$$

gdzie: p - oznacza numer komputera, t(p) - oznacza numer ostatniej kolumny na komputerze p-tym.

Zależność (15) przedstawia symbolicznie sposób wykonania tej operacji, korzystając z wskaźników wierszy, kolumn macierzy R,S, tak jakby znajdowały się one na jednym komputerze. Wymnożenie to powoduje, iż na każdym z komputerów powstaje macierz o

wymiarach identycznych jak wyjściowa macierz A [nxn]. W celu otrzymania macierzy A_i zal. (6) należy przesłać te częściowe wyniki do komputera sterującego i tam zsumować:

$$a_{j,m} = \sum_{p=1}^{P} a_{j,m}^{p} \tag{16}$$



Rys. 1. Algorytm rozwiązania szeregowego
Fig. 1. Non parallel algorithm

Uwagi: (*) - oznacza, iż przekroczona została maksymalna liczba iteracji procesu QR (ograniczenie czasowe),

(**) - może być spełniony, jeśli wszystkie wartości własne macierzy są liczbami rzeczywistymi. Wówczas elementy poddiagonalne dążą do zera (przy braku wielokrotnych wartości własnych).

5. Opis wykorzystywanych mechanizmów rozwiązywania zadania

Do rozwiązania wyżej opisanego zagadnienia wykorzystano mechanizmy zdalnego wywołania procedur (RPC) oraz mechanizmy sygnałów umożliwiających równoległą realizację programu. Idea RPC polega na wykorzystaniu odległych komputerów do realizacji pewnych obliczeń. Mechanizmy zastosowano w systemach sieciowych korzystając z modelu klient/serwer. Komputer lokalny (klient) żąda wykonania usługi od komputera zdalnego (serwer) poprzez wysłanie komunikatu przez sieć. Zaletą RPC jest to, że wywoływane procedury są wykonywane w przestrzeni adresowej odległych komputerów (serwerów). Równoległe wykorzystanie kilku komputerów do realizacji złożonych zadań może przynieść duże korzyści czasowe. Główną trudnością podczas wykorzystania tych mechanizmów jest jednak to, że wywoływane przez nie usługi są realizowane asynchronicznie. Oznacza to, że komputer wywołujący pewne usługi (klient) czeka na ich realizację przez odległy komputer (serwer). Taki sposób wykorzystania tych mechanizmów nie pozwala na równoległe przetwarzanie danych, co jest celem naszych badań. Należy więc spróbować wykorzystać te mechanizmy w sposób synchroniczny. Polega to na tym, że program wywołujący zdalną procedurę nie czeka na jej realizację, lecz przechodzi do wykonywania następnej instrukcji. Wynik wywoływanej procedury jest odbierany później. Można to realizować stosując dodatkowe mechanizmy synchronizujące (takie jak semafory, pamięć dzielona, komunikaty) lub stosując wywołania nieblokujące. Wywołania takie zapewniają równoległe przetwarzanie danych bez konieczności wykorzystywania dodatkowych mechanizmów synchronizujących, co skraca czas obliczeń. W wywołaniach nieblokujących nie ma odpowiedzi od serwera, tak więc klient po wysłaniu rozkazu wykonania odpowiedniej procedury i ewentualnie danych niezbędnych do tego żądania przechodzi do wykonania następnej czynności. Stosowanie wywołań nieblokujących jest możliwe poprzez zerowanie czasu oczekiwania klienta. Należy więc zerować wartości struktury timeval przekazywanej do funkcji clnt_control() oraz zerować parametr timeout przekazywany funkcji clnt_call().

Przykład:

```
/* Program klienta */
.....
CLIENT *klient;
struct timeval tv;
.....
tv.tv_sec = 0;
tv.tv_usec = 0;
clnt_control (klient, CLSET_TIMEOUT, &tv);
.....
/* Program xxx_clnt.c */
.....
static struct timeval NOTIMEOUT = {0, 0};
.....
clnt_call (klient, PROCNAME, xdr_data, parametr, xdr_void, &result, NOTIMEOUT);
.....
```

Przy stosowaniu tej metody serwer nie może zwracać żadnej wartości, należy więc jako wynik odległej procedury przekazać wartość NULL. W definicji procedury w pliku xxx.x należy zdefiniować tę procedurę jako: *void procname() = nr;*

Sygnały natomiast wykorzystano w celu synchronizacji procesów uruchamianych na komputerze klienta i serwera. Stosując wywołania nieblokujące, klient przechodzi do wykonania następnych czynności, kontynuując je do momentu, gdy będą mu potrzebne wyniki wywoływanych procedur. W dotychczasowych opracowaniach stosowano aktywne czekanie polegające na ciągłym sprawdzaniu powstania pliku lub obszaru pamięci dzielonej. Wadą takiej metody jest oczywisty i niepotrzebny wzrost obciążenia procesora komputera lokalnego, powodujący zwolnienie działania komputera, a tym samym całej sieci. Alternatywnym rozwiązaniem jest stosowanie sygnałów, które można przekazać między procesami. Komputer klienta po wywołaniu wszystkich odległych procedur przechodzi w stan zawieszenia wywołując funkcję *pause()*. Powrót z tej funkcji jest możliwy jedynie po nadejściu sygnału pobudki SIGALRM. W celu wykorzystania sygnałów należy na początku programu zdefiniować reakcję programu na oczekiwany sygnał za pomocą funkcji *signal()*. Wysłanie natomiast wybranego sygnału do określonego procesu zapewnia funkcja *kill()*. W naszym przypadku w programie klienta należy definiować: ***signal*** (SIGALRM, pobudka());

Funkcja *pobudka()* jest funkcją użytkownika wykonywaną po nadejściu sygnału pobudki SIGALARM. Zawiera najczęściej informację dla użytkownika potwierdzającą nadejście sygnału. W naszym przypadku jest funkcją pustą: ***void pobudka()*** {};

Wysłanie sygnału pobudki jest inicjowane przez program serwera po wykonaniu wszystkich obliczeń. Wywołanie funkcji ma postać: *kill (idpr, SIGALRM);* gdzie *idpr* jest identyfikatorem procesu odbierającego sygnału.

6. Opis rozwiązania

W celu rozwiązania zadania wykorzystano dwa serwery. Jeden z nich został uruchomiony na odległym komputerze i zapewnił wykonywanie procedur obliczeniowych związanych z wyznaczeniem wartości własnych macierzy. Drugi serwer został uruchomiony na komputerze lokalnym, zapewniając przejście przez kolejne etapy rozwiązania. Takich etapów jest 7, a ich opis przedstawimy później. Rola komputerów włączonych do rozwiązywania zadania jest więc podwójna. W zależności od realizowanego etapu, mogą one grać rolę serwera udostępniającego usługę lub klienta żądającego realizacji zadania. Realizowany algorytm rozwiązywania zadania cechuje się dużym sprzężeniem komputerów, spowodowanym dużą liczbą przesłań między komputerami. Liczba tych przesłań (połączeń, L_p) jest kilkakrotne większa od rozmiaru macierzy. Poszczególne etapy realizowanego algorytmu przedstawiają się następująco:

Etap I: polega na szeregowym przygotowaniu i wysłaniu danych początkowych do odległych komputerów, liczba połączeń jest równa liczbie wykorzystywanych komputerów (L_k). Są to wywołania nieblokujące $L_p = L_k$.

Etap II: polega na wyborze kolejnego komputera i wywołaniu na nim procedury obliczenia wartości cos i sin ze wzoru (7). $L_p = 1$. Są to wywołania blokujące.

Etap III: polega na wysyłaniu do wszystkich odległych komputerów otrzymanych wartości cos i sin z etapu II i wykonaniu procedury mnożenia odpowiednich wierszy macierzy podstawowej i macierzy obrotów Givensa. Po czym komputer lokalny przechodzi do stanu zawieszenia czekając na sygnał pobudki. $L_p = L_k$. Są to wywołania nieblokujące.

Etap IV[1]: wykonywany jest na odległych komputerach, które po wykonaniu mnożeń z etapu poprzedniego stają się klientami, wywołując na lokalnym komputerze (który staje się serwerem) procedurę wysyłającą sygnał pobudki oczekiwany przez komputer klienta. W tym celu procesy te przy użyciu funcji *system()* wywołują zewnętrzny program, który przy użyciu mechanizmów RPC wysyła

[1]Etapy III i IV są wykonywane tyle razy, ile wynosi rozmiar macierzy (R_m).

do odległego komputera[2] sygnał SIGALRM. $L_p = L_k$. Są to połączenia nie.-blokujące.

Etap V: polega na wysłaniu do odległych komputerów rozkazu mnożenia macierzy podstawowej i macierzy obrotów (mnożenie końcowe). Po czym komputer lokalny przechodzi w stan zawieszenia czekając na sygnał pobudki. $L_p = L_k$. Są to połączenia nieblokujące.

Etap VI: jest to powtórzenie etapu IV, gdyż polega na wysyłaniu sygnału pobudki do komputera klienta, który staje się serwerem po wykonaniu operacji mnożeń końcowych. $L_p = L_k$. Są to połączenia nieblokujące.

Etap VII: polega na szeregowym przesłaniu do wszystkich serwerów rozkazu wysłania danych wynikowych i ich sumowaniu, a następnie wyświetleniu na komputerze lokalnym. Są to wywołania blokujące. $L_p = L_k$.

Sumując liczby połączeń wskazane w kolejnych krokach otrzymamy:

$$L_p = 2 * R_m * L_k + 4 * L_k + 1$$

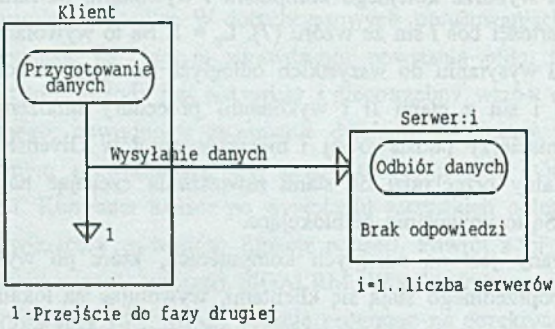
gdzie:

L_p: *Liczba połączeń*

R_m: *Rozmiar macierzy*

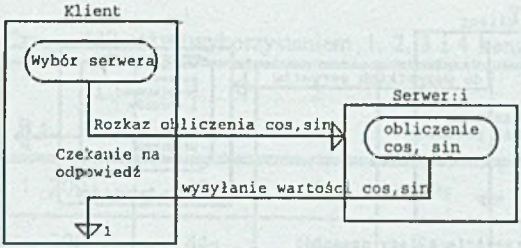
L_k: *Liczba komputerów*

Niżej przedstawione rysunki obrazują wszystkie omawiane etapy.



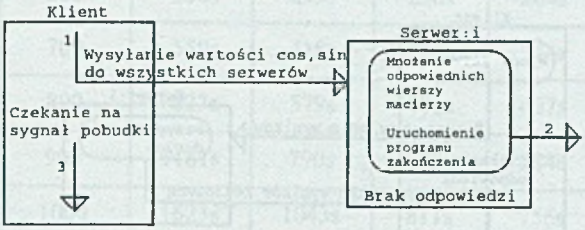
Rys. 2. Faza pierwsza
Fig. 2. First phase

[2]Jest to komputer lokalny (klient) z etapu poprzedniego, który staje się komputerem odległym (serwerem).



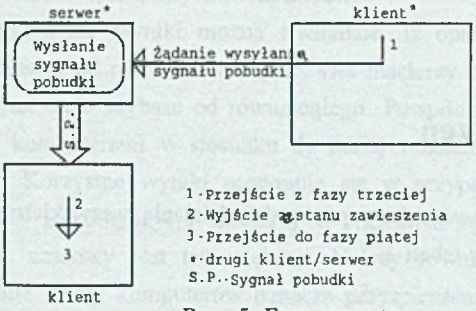
1-Przejście do fazy trzeciej

Rys. 3. Faza druga
Fig. 3. Second phase



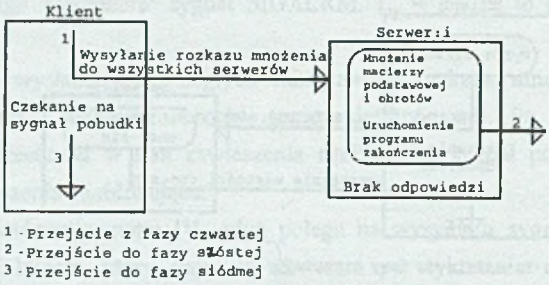
1-Przejście z fazy drugiej
2-Przejście do fazy czwartej
3-Przejście do fazy piątej

Rys. 4. Faza trzecia
Fig. 4.Third phase

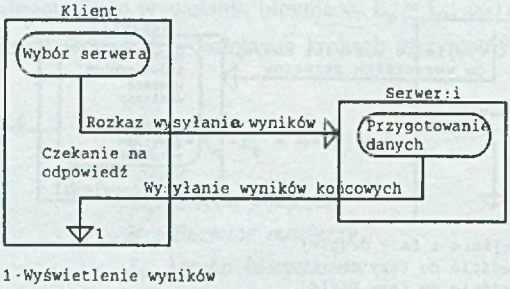


1-Przejście z fazy trzeciej
2-Wyjście z stanu zawieszenia
3-Przejście do fazy piątej
*-drugi klient/serwer
S.P.-Sygnał pobudki

Rys. 5. Faza czwarta
Fig. 5. Fourth phase



Rys. 6. Faza piąta
Fig. 6. Fifth phase



Rys. 7. Faza siódma
Fig. 7. Seventh phase

7. Wyniki obliczeń

Po uruchomieniu i sprawdzeniu poprawności działania wszystkich modułów uzyskane wyniki przedstawiono w tabeli 1.

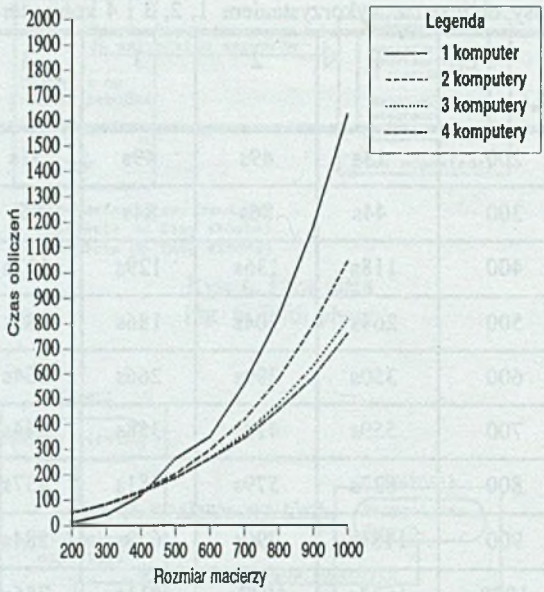
Tabela 1

Czasy obliczeń z wykorzystaniem 1, 2, 3 i 4 komputerów

R_m \ L_k	1	2	3	4
200	13s	49s	49s	51s
300	44s	86s	84s	85s
400	118s	136s	129s	133s
500	264s	204s	186s	189s
600	350s	299s	266s	264s
700	559s	418s	358s	344s
800	827s	579s	481s	457s
900	1181s	790s	628s	584s
1000	1623s	1043s	811s	756s

Tabela ta zawiera wyniki czasowe obliczeń wartości własnych macierzy z wykorzystaniem 2, 3 i 4 komputerów oraz wyniki szeregowego rozwiązania w kolumnie pierwszej. Wszystkie wyniki stanowią średnią kilkakrotnego wykonania zadania realizowanego z wykorzystaniem tych samych komputerów.

Analizując otrzymane wyniki można stwierdzić, iż opłacalność wykorzystywanego algorytmu jest zależna od rozmiaru macierzy. Dla macierzy o małym rozmiarze szeregowe rozwiązanie jest dużo szybsze od równoległego. Powodowane jest to dużym kosztem przesłań między komputerami w stosunku do liczby mnożeń wykonywanych przez odległe komputery. Korzystne wyniki otrzymuje się w przypadku macierzy o rozmiarze większym od 500. Dekompozycja macierzy o większym rozmiarze trwa odpowiednio dłużej, lecz zysk czasowy jest też większy. Należy również zwrócić uwagę, że nie zawsze zwiększenie liczby komputerów oznacza przyspieszenie obliczeń. Wykonanie dekompozycji dla macierzy o rozmiarze 500 z wykorzystaniem czterech komputerów jest jednocześnie szybsze niż dla przypadku szeregowego lub z wykorzystaniem dwóch komputerów, ale przy tym jest wolniejsze niż z wykorzystaniem trzech komputerów. Niżej przedstawiony wykres obrazuje uzyskane wyniki.



Rys. 8. Zależność czas obliczeń/rozmiar macierzy
Fig. 8. Solving time/matrix size dependence

Dokładniejszą analizę można przeprowadzić testując czasy wykonywania poszczególnych etapów rozwiązywania zadania. W tym celu przedstawimy otrzymane czasy dla wybranych rozmiarów macierzy.

Tabela 2
Czasy wykonywania poszczególnych etapów rozwiązywania zadania dla macierzy o rozmiarze 200x200

Czynność / L_k	2	3	4
Przygotowanie danych	0.16s	0.20s	0.25s
Rozsyłanie danych	0.30s	0.29s	0.33s
Obliczenie cos, sin, mnożenie wierszy	39.70s	39.8s	39.9s
Mnożenia końcowe	5.30s	3.70s	2.90s
Zbieranie wyników	4.50s	4.80s	6.50s

Tabela 3

Czasy wykonywania poszczególnych etapów rozwiązywania zadania dla macierzy o rozmiarze 500x500

	2	3	4
Przygotowanie danych	1.8s	1.50s	1.55s
Rozsyłanie danych	2.8s	1.80s	1.80s
Obliczenie cos, sin i mnożenie wierszy	99.7s	100.15s	99.8s
Mnożenia końcowe	80.9s	54.50s	41.5s
Zbieranie wyników	19.9s	30.20s	40.5s

Tabela 4

Czasy wykonywania poszczególnych etapów rozwiązywania zadania dla macierzy o rozmiarze 1000x1000

	2	3	4
Przygotowanie danych	4.77s	4.77s	4.95s
Rozsyłanie danych	7.26s	7.27s	8.17s
Obliczenie cos, sin i mnożenie wierszy	200.8s	200.7s	200.5s
Mnożenia końcowe	653.17s	450.72s	343.85s
Zbieranie wyników	83.19s	128.25s	177.48s

Jak wynika z wyżej przedstawionych tabel, przy małych rozmiarach macierzy (200x200) wszystkie etapy związane z dekompozycją macierzy trwają tym dłużej, im większa jest liczba wykorzystanych komputerów, co jest sprzeczne z wcześniejszymi założeniami (im więcej wykorzystanych komputerów, tym krótszy czas rozwiązywania zadania). Wynika to ze stosunku liczby zdalnie wykonywanych operacji arytmetycznych do liczby połączeń między komputerami. Jedyny niewielki zysk otrzymuje się podczas wykonywania mnożeń końcowych, gdzie liczba operacji jest odpowiednio większa.

Przy większym rozmiarze (500x500) zauważa się pewne wahania, gdyż czasy połączeń zaczynają się równać czasom obliczeń. Oznacza to, że zwiększenie liczby komputerów nie zawsze przyspiesza rozwiązanie zadania. Zdecydowany natomiast zysk otrzymuje się jak poprzednio w ostatniej fazie rozwiązania. Przy takich rozmiarach zysk czasowy jest zdecydowanie zależny od liczby wykorzystywanych komputerów.

Przypadek macierzy o dużych rozmiarach (1000x1000) jest całkowicie zgodny z założeniami. Im więcej komputerów wykorzystujemy, tym większy jest zysk czasowy. Czasy wykonywanych zdalnie operacji przeważają nad czasami połączeń, co oznacza, że zmniejszenie liczby tych operacji poprzez zwiększenie liczby komputerów ma pozytywny wpływ na ostatecznie uzyskany czas.

Ogólnie, zmiany poszczególnych składowych czasu obliczeń scharakteryzować można następująco:

- Przy małych rozmiarach prawie we wszystkich etapach zauważa się wzrost czasu wykonywanej czynności proporcjonalny do liczby wykorzystanych komputerów, zaś niewielki zysk występuje jedynie podczas wykonywania mnożeń końcowych.
- Przy średnich rozmiarach zauważa się pewnego rodzaju wahania w zależności od liczby wykorzystanych komputerów. Najlepsze wyniki uzyskuje się z wykorzystaniem trzech komputerów. Coraz większy jest zysk podczas wykonywania mnożeń końcowych.
- Przy dużych rozmiarach pierwsze etapy są prawie stabilne, etap III wykonuje się coraz szybciej, a zdecydowany zysk otrzymuje się podczas wykonywania mnożeń końcowych, gdzie liczba operacji jest rzędu n^2. Ostatni etap natomiast trwa coraz dłużej z powodu wzrostu rozmiaru danych.

8. Analiza wyników

Niżej przedstawione tabele i wykresy obrazują charakter zmian w procentach w zależności od rozmiaru macierzy i liczby wykorzystywanych komputerów dla poszczególnych etapów rozwiązania zadania. Pozwala to ocenić wpływ każdego etapu na uzyskane wyniki. Analizując poszczególne etapy można stwierdzić, że na wzrost czasu rozwiązania duży wpływ mają operacje związane z przygotowaniem i rozsyłaniem danych, a szczególnie operacje związane ze zbieraniem wyników końcowych. Zyskać czas można natomiat na operacjach związanych z etapem "Mnożenia końcowe", w którym wykonują się intensywne operacje numeryczne. W miarę zwiększenia rozmiaru macierzy i liczby wykorzystanych komputerów zysk czasowy związany z tym etapem jest coraz większy, a straty związane z pozostałymi etapami są coraz mniejsze. Dlatego opłacalność równoległego rozwiązywania zadania jest ściśle związana z rozmiarem macierzy.

Tabela 5

Wyniki uzyskane dla macierzy o rozmiarze 200

	2	3	4
Przygotowanie danych	-	25%	51%
Rozsyłanie danych	-	-2%	9%
Obliczenie cos, sin i mnożenie wierszy	-	0%	0%
Mnożenia końcowe	-	-29%	-45%
Zbieranie wyników	-	+7%	46%

Tabela 6

Wyniki uzyskane dla macierzy o rozmiarze 500

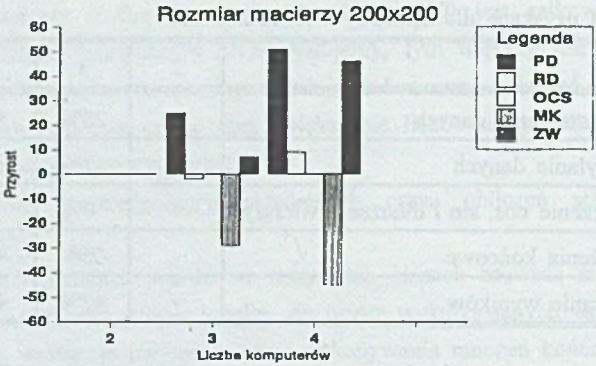
	2	3	4
Przygotowanie danych	-	37%	43%
Rozsyłanie danych	-	-14%	-12%
Obliczenie cos, sin i mnożenie wierszy	-	0.2%	0.09%
Mnożenia końcowe	-	-32%	-48%
Zbieranie wyników	-	51%	103%

Tabela 7

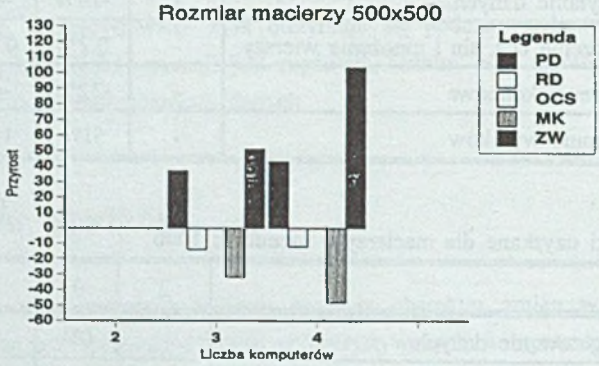
Wyniki uzyskane dla macierzy o rozmiarze 1000

	2	3	4
Przygotowanie danych	-	0%	3%
Rozsyłanie danych	-	0%	12%
Obliczenie cos, sin i mnożenie wierszy	-	0%	0%
Mnożenia końcowe	-	-31%	-47%
Zbieranie wyników	-	54%	113%

We wszystkich tabelach wyniki są podawane względem uzyskanych czasów wykonywania z wykorzystaniem dwóch komputerów. Dlatego pierwsza kolumna nie zawiera żadnych wyników. Rozwiązanie szeregowe nie zawiera takich etapów, a symulacja takiego rozwiązania (obliczenie wartości własnych macierzy zdalnie na jednym komputerze) zawierałaby czasy narzucone przez komunikowanie się komputera odległego z komputerem sterującym.

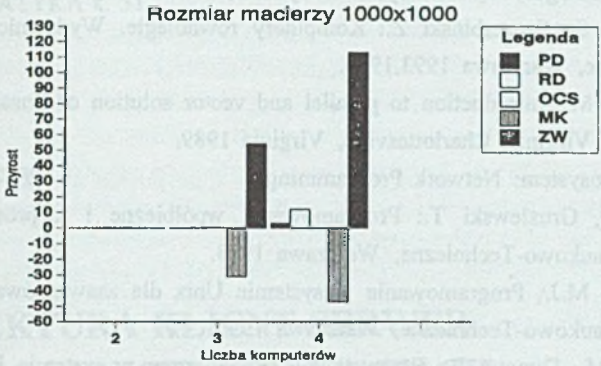


Rys.9. Przyrost czasowy poszczególnych etapów rozwiązywania zadania w stosunku do L_k, R_m=200
Fig.9. Time growth of each algorithm phase, matrix size 200x200



Rys.10. Przyrost czasowy poszczególnych etapów rozwiązywania zadania w stosunku do L_k, R_m=500
Fig.10. Time growth of each algorithm phase, matrix size 500

Występujące skróty w legendach rysunków 9, 10 i 11 oznaczają:
PD - Przygotowanie Danych, RD - Rozsyłanie Danych, MK - Mnożenia Końcowe, OCS - Obliczenie Cos, Sin i mnożenie wierszy, ZW - Zbieranie Wyników.



Rys.11. Przyrost czasowy poszczególnych etapów rozwiązywania zadania w stosunku do L_k, R_m=1000
fig.11. Time growth of each algorytm phase, matrix size 1000x1000

9. Wnioski

Reasumując, można stwierdzić, że stosowanie nieblokujących mechanizmów zdalnego wywołania procedur jest dobrym i szybkim narzędziem, pozwalającym na implementację rownoległych aplikacji. Wielkość zysku czasowego zależy jednak od wielu czynników, takich jak obciążenie sieci, rozmiar rozsyłanych danych, liczba wykorzystanych komputerów. Najważniejszym czynnikiem wpływającym na uzykane wyniki jest jednak stosunek liczby połączeń do liczby wykonywanych operacji na odległe komputery. Im większa liczba wykonywanych operacji na serwerach przy stałej lub malejącej liczby połączeń, tym większy zysk. Jest to podstawą do dalszych badań mających na celu określenie granicy opłacalności wykorzystania mechanizmów zdalnego wywołania procedur do obliczeń rownoległych.

LITERATURA

[1] Kundur P., Porretta B., Rogers G.J, Wong D.Y.: Eigenvalue analisis of very large power systems. IEEE Transactions on Power Systems, Vol.3, No.2, May 1988 pp. 472-480.

[2] Wanat K.: Algorytmy numeryczne. Wyd.Dir, Gliwice 1993.

[3] Kozielski S., Szczerbiński Z.: Komputery równoległe. Wydawnictwo Naukowo-Techniczne, Warszawa 1993,1994.

[4] Ortega J. M.: Introduction to parallel and vector solution of linear systems. University of Virginia, Charlottesville, Virginia 1989.

[5] Sun Microsystem: Network Programming.

[6] Weiss Z., Grużlewski T.: Programowanie wpółbieżne i rozproszone. Wydawnictwo Naukowo-Techniczne, Warszawa 1993.

[7] Rochkind M.J.: Programowanie w systemie Unix dla zaawansowanych. Wydawnictwo Naukowo-Techniczne, Warszawa 1993.

[8] Gabassi M., Dupouy B.: Przetwarzanie rozproszone w systemie UNIX. LUPUS, Warszawa 1995.

Recenzent: Doc.dr hab. inż. Tadeusz Czachórski

Wpłynęło do Redakcji 16 października 1995 r.

Abstract

The paper presents an example of using the remote procedure call (RPC) in Unix network system, to solve in parrallel big eigensystems. To shorten the solution time, we use the nonblocking procedure call in combination with signals for inter process synchronization. Table 1 contains the results of the experiment. The efficiency of the used algorithm depends on the solved matrix size. For small matrices, parallel solution is not effective. Favourable results may be obtained for matrices whose size is equal or greater than 500x500. The obtained results depend not only on matrix size but also on the number of used remote computers. Tables 2-4 present the execution time of algorithm's fragments for different matrix size (respectively 200x200, 500x500 and 1000x1000) using 2, 3 and 4 remote machines. The solving time/matrix size depencency presents fig.8 . Fig.2-7 present 6 different phases of the used algorithm. Finally, fig.9-11 present the growth of the execution time growth for algorithm's phases as a function of different matrix sizes and different number of remote computers.