ZESZYTY NAUKOWE·POLITECHNIKI ŚLĄSKIEJ 1997
Seria: INFORMATYKA z. 32 Nr kol. 1356

Hafed ZGHIDI
Politechnika Śląska, Instytut Informatyki

WPŁYW METODY WYKORZYSTANIA MECHANIZMÓW ZDALNEGO WYWOŁANIA PROCEDUR NA CZAS REALIZACJI RÓWNOLEGŁYCH ZADAŃ

Streszczenie. Niniejszy artykuł przedstawia różne metody wykorzystania mechanizmów zdalnego wywołania procedur do realizacji równoległego wyznaczania wartości własnych macierzy o dużych rozmiarach. Wpływ każdej z przedstawionych metod na czas wykonywania zadania jest tematem dyskusji o wadach i zaletach każdej z nich.

THE INFLUENCE OF DIFFERENT METHODS OF REMOTE PROCEDURE CALL MECHANISMS ON PARALLEL COMPUTING TIME

Summary. The paper shows different methods to use Remote Procedure Call mechanisms for parallel computing. A discussion is provided about each of them. The time of parallel solution of large eigensystems using each of these methods is different, so a discussion about the efficiency of each method is provided.

1. Wstęp

Mechanizmy zdalnego wywołania procedur (RPC) można wykorzystać do realizacji obliczeń równoległych. Pozwalają one na wykorzystanie mocy obliczeniowej oddalonych komputerów celem realizacji pewnych zadań. Mechanizmy RPC oparte są na architekturze klient/serwer, dlatego znalazły one zastosowanie w sieciowych systemach komputerowych, gdzie istnieje możliwość włączania większej liczby komputerów do realizacji pojedynczego

zadania. Mechanizmy te można wykorzystać w różnorodny sposób. Wybór właściwego sposobu ich wykorzystania zależy od użytkownika, zagadnienia i algorytmu rozwiązania zadania. Celem niniejszej pracy jest zbadanie wpływu metody wykorzystania mechanizmów RPC na czas realizacji zadania. Opisane w niej badania zostały wykonane w środowisku sieciowym z systemem operacyjnym Unix, na stacjach roboczych Sun o porównywalnej mocy obliczeniowej.

2. Sposoby wykorzystania RPC

Działanie RPC polega na wysłaniu przez komputer lokalny (klient), przez sieć do wybranego zdalnego komputera (serwer), komunikatu żądania wykonania pewnych usług (procedur) i czekanie na ich realizację [3][5][6][8]. Zadania te są realizowane w przestrzeni adresowej komputera oddalonego, co zwalnia komputer lokalny od ich realizacji. Jest to jednak asynchroniczny wariant wykorzystania RPC, który wyklucza równoległą realizację zadań. Dlatego wykorzystanie RPC do równoległych zadań musi się odbywać w wariancie synchronicznym. Wiąże się to często ze stosowaniem dodatkowych mechanizmów systemowych, takich jak: pamięć dzielona, sygnały, semafory, komunikaty itd. celem synchronizacji procesów odbywających się na komputerze klienta i serwerach. Mechanizmy te powinny zapewnić procesowi klienta przejścia przez kolejne etapy rozwiązania zadania. Wykorzystanie dodatkowych mechanizmów ma ujemny wpływ na końcowy czas realizacji zadania. Ich zastosowanie wiąże się zawsze z pewnym narzutem czasowym. Należy więc w miarę możliwości z nich zrezygnować.

Schematy metod wykorzystania RPC przedstawiają następne punkty.

2.1. Wywołania blokujące

Taki sposób wykorzystania RPC nie pozwala na równoległą realizację zadań, gdyż komputer lokalny (klient) wywołujący zdalne usługi czeka na ich realizację poprzez serwer. Taka metoda jest jednak często stosowana podczas inicjacji procesów oddalonych. Można ją wykorzystać do wysyłania danych początkowych, zmiennych globalnych itd. (rys. 1).

2.2. Wywołania blokujące, pamięć dzielona i powłoka zleceń

Serwer po otrzymaniu zadania zapisuje je do obszaru pamięci dzielonej i uruchamia zewnętrzny program do jego realizacji w tle - korzystanie z powłoki zleceń (Shell) (funkcja exec() lub system() z &) - i nie czekając na jego zakończenie wysyła potwierdzenie odbioru zadania do klienta. Klient przechodzi wtedy do realizacji następnej czynności. Klient po zainicjowaniu wszystkich zdalnych zadań czeka na ich realizację. Po zakończeniu działania programy uruchamiane zdalnie na serwerach wysyłają do klienta informacje o zakończeniu zadania wykorzystując mechanizmy RPC. Klient przechodzi wtedy do realizacji następnego kroku algorytmu lub do odbioru wyników obliczeń (rys. 2).

2.3. Wywołania nieblokujące i powłoka zleceń

Klient po wysłaniu zadania nie czeka na potwierdzenie ich odbioru od serwera, lecz przechodzi do wykonywania następnej czynności. Serwer nie mając obowiązku potwierdzenia otrzymania zadania przystępuje do jego realizacji. Po jego zakończeniu przesyła za pomocą RPC - uruchamiając program zewnętrzny (korzystanie z Shella: funkcja exec() lub system() z &) - informację o zakończeniu działania do klienta. Klient uruchamia wtedy procedurę odbioru wyników także korzystając z mechanizmów RPC lub przechodzi do wykonania następnego kroku algorytmu (rys. 3).

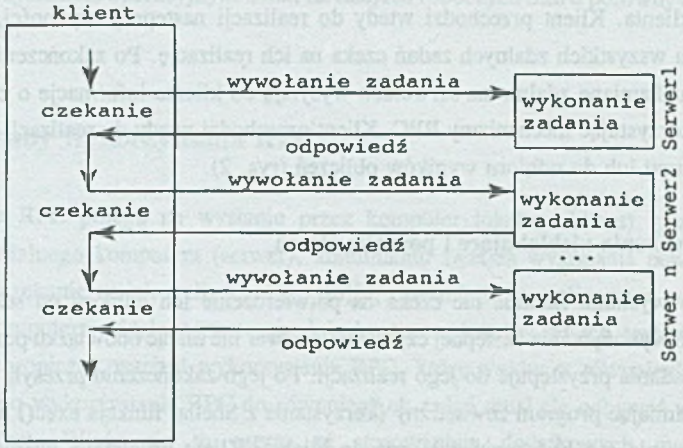
2.4. Wywołania nieblokujące bez powłoki zleceń (klient/serwer)

Taki sposób wykorzystania RPC różni się od poprzedniego tym, że serwer po wykonaniu zadania wysyłając informację do klienta nie wywołuje zewnętrznego programu, lecz odgrywa rolę klienta wywołując odpowiednie procedury RPC. W ten sposób komputer zdalny pełni dwie funkcje kolejno serwera i klienta, a komputer lokalny kolejno funkcje klienta i serwera (rys. 4).

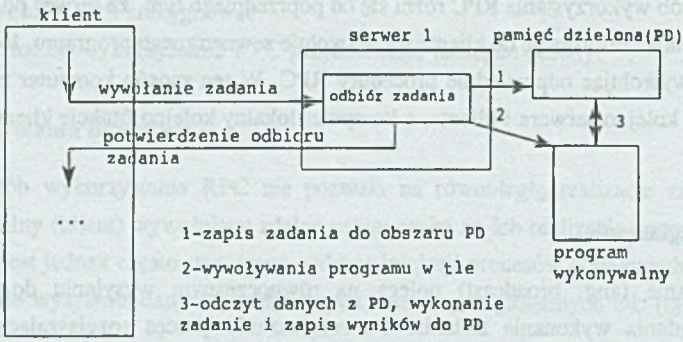
2.5. Rozgłaszanie

Rozgłaszanie (ang. broadcast) polega na równoczesnym wysyłaniu do wszystkich serwerów żądania wykonania zadania. Po rozgłaszaniu proces rozgłaszający czeka na zakończenie wykonania zdalnych procedur. Po nadejściu odpowiedzi od któregokolwiek z serwerów proces rozgłaszający przechodzi do realizacji ciągu instrukcji zdefiniowanego przez użytkownika, który może korzystać z otrzymanej odpowiedzi. Powrót z rozgłaszania następuje w momencie, kiedy spełniony zostaje warunek logiczny również podany przez

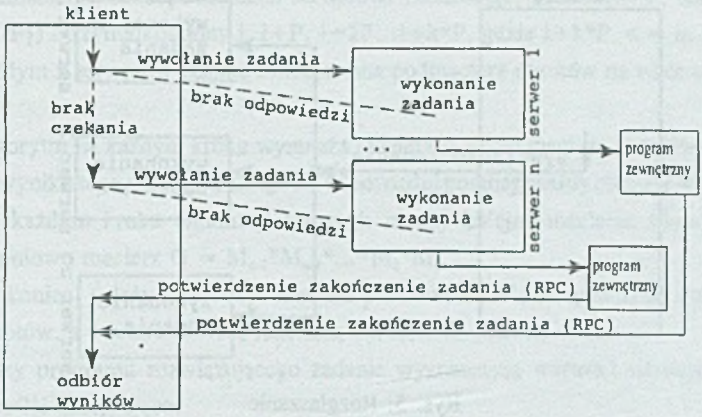
użytkownika. Wszystkie odpowiedzi nadchodzące od oddalonych serwerów po spełnieniu tego warunku są zignorowane. Jeśli natomiast warunek nie zostaje spełniony, to po upływie określonego przedziału czasu proces rozgłaszający ponownie wykonuje rozgłaszanie tego samego komunikatu do wszystkich procesów obsługujących (rys. 5).



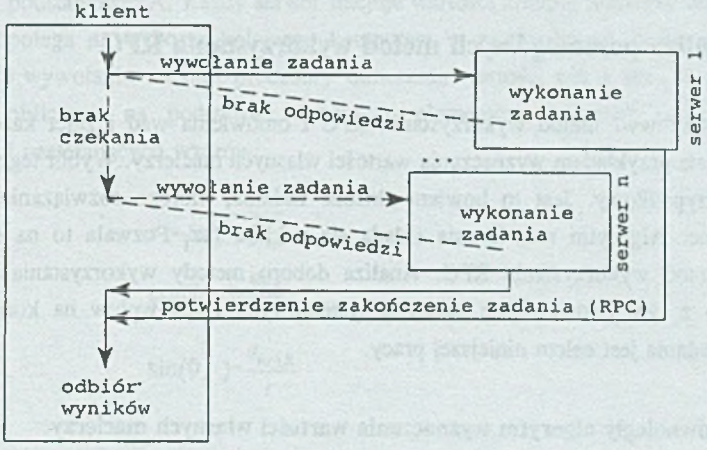
Rys. 1. Wywołania blokujące
Fig. 1. Blocking calls



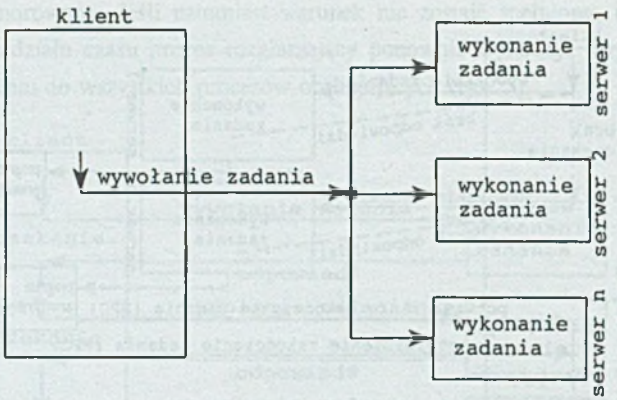
Rys. 2. Wywołania blokujące, pamięć dzielona i Shell
Fig. 2. Blocking calls, shared memory and Shell



Rys. 3. Wywołania nieblokujące i Shell
Fig. 3. Nonblocking calls and Shell



Rys. 4. Wywołania nieblokujące (klient/serwer)
Fig. 4. Nonblocking calls (client/server)



Rys. 5. Rozgłaszanie
Fig. 5. Broadcast

3. Analiza poszczególnych metod wykorzystania RPC

Do analizy ww. metod wykorzystania RPC i omówienia wad i zalet każdej z nich posłużymy się przykładem wyznaczania wartości własnych macierzy. Wybór tego problemu nie jest przypadkowy. Jest to bowiem złożone zadanie, którego rozwiązanie jest dość czasochłonne. Algorytm rozwiązania składa się z kilku faz. Pozwala to na stosowanie różnych metod wykorzystania RPC. Analiza doboru metody wykorzystania RPC oraz związanych z tym dodatkowych narzędzi systemowych i ich wpływ na końcowy czas realizacji zadania jest celem niniejszej pracy.

3.1. Równoległy algorytm wyznaczania wartości własnych macierzy

Równoległy algorytm wyznaczania wartości własnych macierzy [1][2][4][9] oparty jest na wykorzystaniu przekształcenia QR. Skrótowy opis algorytmu jest następujący:

1. Dane wejściowe stanowi niesymetryczna macierz A[nxn] o elementach rzeczywistych, macierz obrotów Givensa M[nxn], która w początkowej postaci jest jednostkową macierzą diagonalną, oraz sieć komputerowa składająca się z P+1 komputerów, na które składa się P jednostek wykonawczych (serwerów) oraz jedna sterująca (klient).

2. Macierz A należy podzielić wg kolumn w jednostce sterującej i wysłać kolejne podmacierze do odpowiednich serwerów. Zakładając, że n >> P, serwer i (i ∈ {1..P}) otrzyma kolumny i, i+P, i+2P...i+k*P, gdzie i+k*P <= n, k ∈ N. Na każdym z serwerów zostaje zainicjowana podmacierz obrotów na wzór podmacierzy A.
3. Algorytm na każdym kroku wymnaża macierz A przez macierz obrotów Givensa M. W wyniku tego mnożenia zeruje się odpowiedni poddiagonalny element w macierzyA.
4. Na każdym kroku eliminacji algorytm mnoży kolejne macierze obrotów tworząc stopniowo macierz $G = M_{n-1}*M_{n-2}*...*M_2*M_1$.
5. Na koniec, należy wymnożyć macierz podstawową A', otrzymaną po dokonaniu n-1 obrotów, z transponowaną macierzą obrotów G.

Kolejne fazy programu rozwiązującego zadanie wyznaczenia wartości własnych macierzy można określić następująco:

Faza I: polega na szeregowym przygotowaniu i wysłaniu danych początkowych do odległych komputerów. Każdy serwer otrzymuje odpowiednie kolumny macierzy podstawowej A. Każdy serwer inicjuje wartości kolumn macierzy obrotów M.

Faza II: polega na wyborze kolejnego komputera w zależności od kroku algorytmu k i wywołaniu na nim procedury obliczenia wartości cos i sin. Wartości te są obliczane na podstawie wybranych elementów macierzy podstawowej A i następujących wzorów:

$$r = \sqrt{a_{k,k}^2 + a_{k+1,k}^2}$$
$$\cos(\vartheta_k) = \frac{a_{k,k}}{r} \qquad (1)$$
$$\sin(\vartheta_k) = \frac{a_{k+1,k}}{r}$$

gdzie: $a_{k,k}, a_{k+1,k}$ elementy macierzy A dla kroku algorytmu k.

Faza III: polega na wysyłaniu do wszystkich odległych komputerów otrzymanych wartości cos i sin z fazy II i wykonaniu procedury mnożenia odpowiednich wierszy macierzy podstawowej A i macierzy obrotów Givensa G przez macierz obrotów M_k, której elementy $M_{k,k} = M_{k+1,k+1} = \cos(\vartheta)$, $M_{k+1,k} = \sin(\vartheta)$, $M_{k,k+1} = -\sin(\vartheta)$. W wyniku tego mnożenia zeruje się element $a_{k+1,k}$ macierzy A i aktualizuje się

wartości macierzy obrotów G. Po wywołaniu tych procedur komputer lokalny przechodzi do stanu zawieszenia czekając na zakończenie obliczeń.

Faza IV: polega na wysłaniu do odległych komputerów rozkazu mnożenia macierzy podstawowej i macierzy obrotów (mnożenie końcowe). Następnie komputer lokalny przechodzi w stan zawieszenia czekając na zakończenie obliczeń.

Faza V: polega na szeregowym przesłaniu do wszystkich serwerów rozkazu wysłania danych wynikowych, ich sumowaniu i wyświetleniu na komputerze lokalnym.

Każdą z ww. faz można zrealizować z wykorzystaniem RPC stosując dowolną z metod opisanych w paragrafie 3.

3.2. Warianty rozwiązania zadania

Poniżej przedstawimy kilka wariantów rozwiązania powyższego zadania.

3.2.1. Wariant I

Pierwszy wariant rozwiązania zadania polegał na wykorzystaniu wywołań blokujących celem zapewnienia komunikacji między komputerami (metoda 2.2). Do synchronizacji procesów lokalnych i zdalnych oraz przejścia przez kolejne etapy rozwiązania wykorzystano pamięć dzieloną i aktywne czekanie. W tym przypadku realizacja kolejnych faz rozwiązywania zadania wyglądała następująco:

Faza I: serwer po otrzymaniu danych zapisuje je do obszaru pamięci dzielonej i wysyła potwierdzenie ich odbioru do klienta.

Faza II: po określeniu kolejnego kroku rozwiązania klient wysyła do wybranego serwera rozkazy obliczenia wartości cos i sin. Serwer odczytuje macierz zapisaną w obszarze pamięci dzielonej, oblicza wartości cos i sin, które wysyła klientowi.

Faza III: klient wysyła otrzymane wartości cos i sin do kolejnego serwera. Serwer je zapisuje do obszaru pamięci dzielonej, wywołuje program zewnętrzny, który wykonuje mnożenia rownoległe w tle i wysyła potwierdzenie odbioru cos i sin do klienta. Klient po wysłaniu danych do wszystkich serwerów czeka aktywnie na powstanie obszaru pamięci dzielonej z wiadomością o zakończeniu obliczeń. Serwer po zakończeniu obliczeń uruchamia zewnętrzny program, który tworzy u klienta obszar pamięci dzielonej z informacją o zakończeniu obliczeń.

Faza IV: klient wysyła do wszystkich serwerów rozkaz wykonania mnożeń końcowych. Serwer uruchamia zewnętrzny program, który wykonuje obliczenia w tle, i nie czekając na jego zakończenie potwierdza odbiór zadania do klienta. Klient po otrzymaniu wszystkich potwierdzeń czeka aktywnie na zakończenie odległych

programów. Programy uruchamiane zdalnie w tle czytają wartości macierzy z obszaru pamięci dzielonej. Wykonują one obliczenia, zapisują wyniki do tego samego obszaru pamięci dzielonej i wywołują program zewnętrzny, który za pomocą RPC zapisuje do obszaru pamięci dzielonej komputera klienta informację o zakończeniu działania.

Faza V: klient odbiera od kolejnego serwera wyniki końcowe. Każdy serwer kasuje tworzone obszary pamięci dzielonej celem ich ponownego wykorzystania.

W tym wariancie wszystkie wywołania z wykorzystaniem RPC są wywołaniami blokującymi (metoda 2.1).

3.2.2. Wariant II

Drugi wariant rozwiązania zadania polegał na wykorzystaniu wywołań blokujących i nieblokujących celem zapewnienia komunikacji między komputerami. Stosowano również mechanizmy sygnałów do synchronizacji procesów lokalnych i zdalnych oraz do przejścia przez kolejne etapy rozwiązania. Zrezygnowano z pamięci dzielonej i aktywnego czekania. W tym przypadku realizacja kolejnych faz rozwiązywania zadania wyglądała następująco:

Faza I: klient po dekompozycji macierzy podstawowej wysyła kolejne podmacierze do kolejnych serwerów nie czekając na potwierdzenie ich odbioru. Są to wywołania nieblokujące.

Faza II[1]: po określeniu kolejnego kroku rozwiązania klient wysyła do wybranego serwera rozkazy obliczenia wartości cos i sin, i czeka na ich otrzymanie. Są to wywołania blokujące.

Faza III: klient wysyła wartości cos i sin do kolejnego serwera i wywołuje procedury mnożenia nie czekając na potwierdzenie ich odbioru. Są to wywołania nieblokujące. Klient po wysłaniu danych do wszystkich serwerów nie czeka aktywnie na zakończenie obliczeń, lecz przechodzi do fazy zawieszenia czekając na sygnał pobudki. Serwer po zakończeniu obliczeń uruchamia zewnętrzny program, który za pomocą RPC wysyła sygnał pobudki do komputera klienta.

Faza IV: klient wysyła do wszystkich serwerów rozkaz wykonania mnożeń końcowych nie czekając na potwierdzenie odbioru zadania. Są to wywołania nieblokujące. Klient po wysłaniu wszystkich zadań przechodzi do fazy zawieszenia czekając na sygnał

[1]Fazy II i III (we wszystkich wariantach rozwiązania) są wykonywane tyle razy, ile wynosi rozmiar macierzy.

pobudki. Serwer po zakończeniu obliczeń uruchamia zewnętrzny program, który za pomocą RPC wysyła sygnał pobudki do komputera klienta.

Faza V: klient wysyła do każdego serwera rozkaz wysłania wyników końcowych i czeka na ich odbiór. Są to wywołania blokujące.

W tym wariancie wywołania z wykorzystaniem RPC są wywołaniami blokującymi i nieblokującymi (metody 2.1, 2.2 i 2.3).

3.2.3. Wariant III

Trzeci wariant rozwiązania zadania jest zbliżony do wariantu II. Różnica polega na tym, że serwery po zakończeniu obliczeń, wysyłając sygnał pobudki do komputera klienta, nie uruchamiają zewnętrznego programu (przy wykorzystaniu funkcji exec() lub system()), lecz wywołując wewnętrzne procedury zamieniają się w klienta i za pomocą RPC wysyłają sygnał pobudki do komputera klienta, który staje się dla nich serwerem (metoda 2.4).

3.2.4. Wariant IV

Czwarty wariant różni się od wariantu III tym, że w fazie III komputer klienta korzysta z funkcji rozgłaszania do wysyłania wartości cos i sin otrzymanych z fazy II. W ten sposób klient po rozgłaszaniu wartości cos i sin po całej sieci i wywołaniu odpowiednich procedur nie przechodzi w stan zawieszenia i nie czeka na sygnał pobudki, lecz po otrzymaniu potwierdzenia odbioru zadania przechodzi do kolejnego kroku progamu (metoda 2.5).

4. Wyniki obliczeń

Na wstępie należy przypomnieć, że celem naszych badań jest określenie wpływu metody wykorzystania RPC na końcowy czas realizacji zadania. Dlatego inne eksperymenty mające na celu określenie wpływu liczby wykorzystanych komputerów lub rozmiaru macierzy na czas realizacji zadania nie są brane pod uwagę.

I tak do pokazania wpływu każdej z ww. metod na czas realizacji zadania przedstawimy czasy wyznaczania wartości własnych macierzy kwadratowej o rozmiarze 600 z wykorzystaniem 3 komputerów Sun działających w środowisku sieciowym z systemem operacyjnym Unix. Dodatkowo przedstawimy czas szeregowego rozwiązania[1].

[1]Przez szeregowe rozwiązanie zadania rozumie się odrębny algorytm wyznaczania wartości własnych macierzy uruchomiony na jednym komputerze Sun.
m1: metoda 2.1, m2: metoda 2.2 itd.

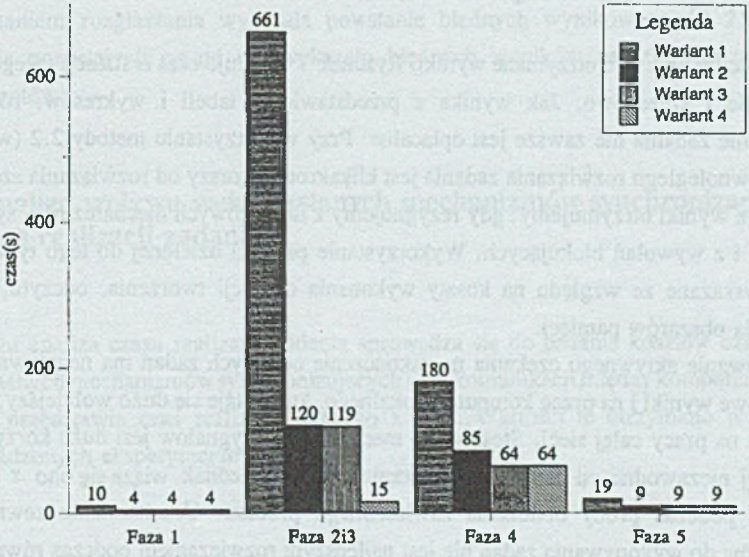
Tabela 1 zawiera czasy realizacji poszczególnych faz rozwiązania zadania dla każdej z opisanych metod. Ponieważ szeregowy algorytm rozwiązywania zadania nie zawiera ww. etapów, wiersz 5 ich nie uwzględnia.

Tabela 1

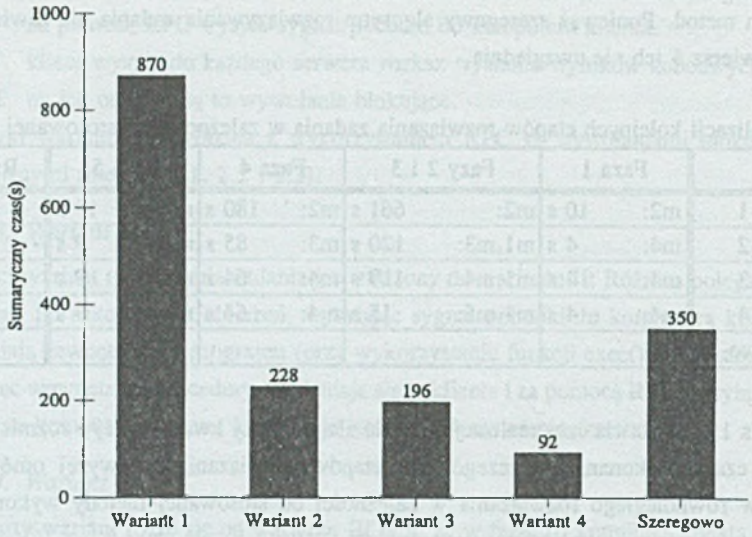
Czas realizacji kolejnych etapów rozwiązania zadania w zależności od stosowanej metody

	Faza 1	Fazy 2 i 3	Faza 4	Faza 5	Razem
Wariant 1	m2: 10 s	m2: 661 s	m2: 180 s	m2: 19 s	870 s
Wariant 2	m4: 4 s	m1,m3: 120 s	m3: 85 s	m1: 9 s	228 s
Wariant 3	m4: 4 s	m1,m4: 119 s	m4: 64 s	m1: 9 s	196 s
Wariant 4	m4: 4 s	m1,m5: 15 s	m4: 64 s	m1: 9 s	92 s
Szeregowo	-	-	-	-	350 s

Tabela 1 przedstawia czas realizacji zadania dla macierzy kwadratowej o rozmiarze 600. Zawiera czasy wykonania poszczególnych etapów rozwiązania dla wyżej omówionych wariantów równoległego rozwiązania w zależności od stosowanej metody wykorzystania mechanizmów zdalnego wywołania procedur.



Rys. 6. Czas realizacji poszczególnych faz algorytmu dla różnych wariantów rozwiązania
Fig. 6. Elapsed time of each algorithm phase for each solution method



Rys. 7. Czas wykonania zadania dla różnych wariantów rozwiązania
Fig. 7. Elapsed time for different solution methods

Rysunek 6 obrazuje otrzymane wyniki. Rysunek 7 obrazuje czas realizacji całego zadania równolegle i szeregowo. Jak wynika z przedstawionej tabeli i wykresów, równoległe rozwiązanie zadania nie zawsze jest opłacalne. Przy wykorzystaniu metody 2.2 (wariant 1) wynik równoległego rozwiązania zadania jest kilkakrotnie gorszy od rozwiązania szeregowego. Dobre wyniki otrzymujemy, gdy rezygnujemy z dodatkowych mechanizmów synchronizujących i z wywołań blokujących. Wykorzystanie pamięci dzielonej do tego typu zadania jest niewskazane ze względu na koszty wykonania operacji tworzenia, odczytu, zapisu i kasowania obszarów pamięci.

Stosowanie aktywnego czekania na zakończenie odległych zadań ma negatywny wpływ na końcowe wyniki i na pracę komputera lokalnego, który staje się dużo wolniejszy, co może się odbić na pracy całej sieci. Stosowanie mechanizmów sygnałów jest dużo korzystniejsze i bardziej niezawodne od aktywnego czekania, niemniej jednak wiąże się ono z pewnymi kosztami podczas próby obudzenia zawieszonego procesu. Uruchamianie zewnętrznych programów do wykonywania zadań nie jest najlepszym rozwiązaniem podczas równoległego wykonywania zadań. Wiąże się to z tym, że wywołanie funkcji systemowych exec() lub system() ponownie inicjuje wykonywany proces na podstawie wskazanego programu [7].

Zmieniany jest wtedy wykonywany program i wymaga to czasu. Unikanie tych wywołań poprzez wykorzystanie mechanizmów RPC jest bardziej opłacalne i dużo szybsze.

Choć czas realizacji pojedynczej operacji tworzenia, zapisu lub odczytu z obszaru pamięci dzielonej, jak i czas uruchamiania programu zewnętrznego lub wysyłanie sygnału pobudki do zawieszanego procesu jest bardzo mały, to przy dużych rozmiarach macierzy podstawowej ich suma staje się duża i końcowy czas realizacji zadania staje się niekorzystny.

Jak wynika z tabeli i wykresów korzystanie z rozgłaszania daje najlepsze wyniki. Jest to zrozumiałe, gdyż rozgłaszanie umożliwia równoczesne wywołanie tej samej procedury na wszystkich odległych procesorach, tym bardziej że w analizowanym zadaniu takich wywołań jest bardzo dużo. Należy jednak podkreślić, że korzystający z rozgłaszania zmuszony jest do stosowania protokołu transmisji UDP/IP. Oznacza to, że rozmiar komunikatu wysyłanego przez sieć, jak i otrzymanej odpowiedzi nie może przekroczyć 8 kB. Po otrzymaniu odpowiedzi lub w przypadku jej braku nie ma możliwości prostego sprawdzania, od którego serwera odpowiedź nadeszła, lub który serwer nie odpowiedział. Powtórne wysyłanie komunikatu do wszystkich serwerów po upływie czasu oczekiwania nie zawsze jest najlepszym rozwiązaniem. Powoduje ono zwykle otrzymanie błędnych wyników i niepotrzebne obciążenie serwerów. Kilkakrotna realizacja wyżej opisanego zadania z wykorzystaniem rozgłaszania wykazała powstanie błędnych wyników wśród 2% prób. Stosowanie pozostałych metod nie wykazało błędnych wyników na 100 realizowanych eksperymentów[1].

5. Analiza wpływu wykorzystanych mechanizmów synchronizacji na czas realizacji zadania

Głębsza analiza czasu realizacji zadania sprowadza się do badania kosztów czasowych wykorzystanych mechanizmów synchronizujących oraz komunikacji między komputerami [7]. Tabela 2 przedstawia czas realizacji każdego z nich. Wartości te otrzymano w wyniku przeprowadzonych eksperymentów.

[1]Ze względu na długi czas realizacji zadania metodą 2 (870s ≈ 15min) ograniczono powtórzenie eksperymentów do 10 prób.

Tabela 2

Czas pojedynczej operacji synchronizującej

Operacja	Czas (ms)
Dostęp do pamięci dzielonej	200
Budzenie procesu	15
Aktywne czekanie	17
Uruchomienie programu zewnętrznego	60

Należy podkreślić, że czas wykonania operacji budzenia procesu jest zależny od obciążenia komputera. Ponieważ zadanie to wykonano w środowisku systemu Unix, w którym liczba aktywnych procesów jest zmienna, otrzymano różne czasy realizacji tej operacji. Tabela 2 zawiera średnią 100 wykonanych eksperymentów. Do określenia wpływu aktywnego czekania na czas realizacji zadania zastosowano dodatkowy program pomocniczy. Okazało się, że jest on porównywalny z czasem budzenia procesu z zastosowaniem sygnałów. Taki sposób synchronizacji jest jednak niekorzystny, gdyż wpływa negatywnie na działanie sieci komputerowej, a szczególnie na procesy odbywające się równocześnie na tej samej stacji roboczej.

Czas realizacji operacji dostępu do pamięci dzielonej zależy od:

- rodzaju operacji (tworzenie/kasowanie obszaru PD, odczyt/zapis do PD),
- rozmiaru danych przeznaczonych do zapisu lub odczytu.

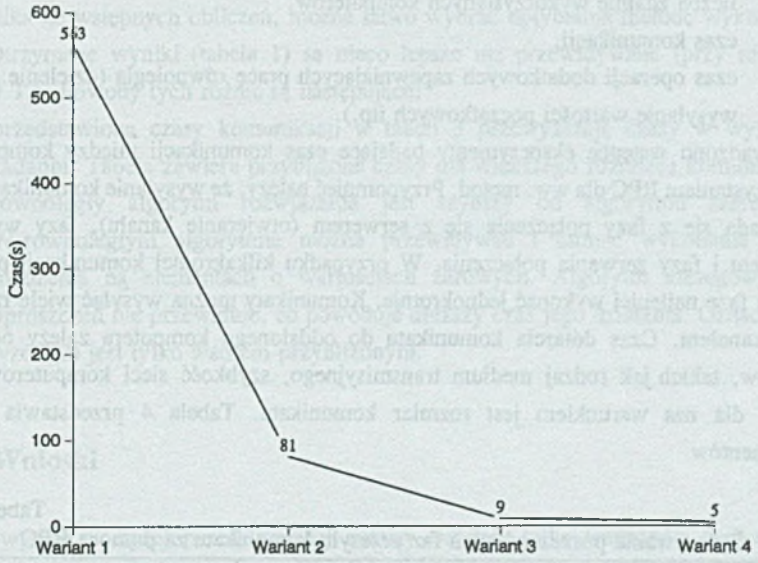
Ponieważ operacje tworzenia i kasowania obszaru pamięci dzielonej realizowane są zwykle jednokrotnie (na początku obliczeń), to ich czas wykonania ma niewielki wpływ na otrzymane wyniki. Operacja zapisu do PD ww. podmacierzy A i M dla rozmiaru macierzy podstawowej równego 600 wynosi 290 ms, odczytu zaś 110 ms. Ponieważ te operacje wykonuje się zwykle parami, można stwierdzić, że średni czas dostępu do PD wynosi 200 ms. Chcąc badać wpływ każdego z mechanizmów na końcowy czas wykonania zadania, należy obliczyć, ile razy został on wykorzystany w każdym z ww. wariantów rozwiązania. Tabela 3 przedstawia wyniki badań. Ostatnia kolumna zawiera sumaryczny czas wykorzystania mechanizmów synchronizujących w zaokrągleniu.

Tabela 3

Liczba wykonania operacji synchronizacji dla każdego wariantu rozwiązania

	Pamięć dzielona	Aktywne czekanie	Sygnały	Uruchomienie programu zewnętrznego	Łączny czas (s)
Wariant 1	2404	601	-	1202	563
Wariant 2	-	-	601	1202	81
Wariant 3	-	-	601	-	9
Wariant 4	-	-	301	-	5

Rysunek 8 obrazuje wyniki otrzymane z tabeli 3.



Rys. 8. Zależność czas synchronizacji/wariant rozwiązania
Fig. 8. Synchronisation time for different solution methods

6. Model przewidywania opłacalności algorytmu równoległego

Podjęcie problematyki rozwiązywania zadań równoległych wiąże się z poświęceniem czasu celem ich realizacji. Jak już wspomniano, nie zawsze jest to opłacalne. Przydatne są więc wstępne obliczenia pozwalające na przewidywania czasu realizacji zadania. Założony przy rozpraszaniu zadań model analityczny można określić następującym wzorem:

$$T_r = S_n + \frac{T_s}{L_c} + T_k + T_d \qquad (2)$$

gdzie:

T_s – czas szeregowego rozwiązania zadania,

T_r – czas równoległego rozwiązania zadania,

S_n – czas synchronizacji,

L_c – liczba zdalnie wykorzystanych komputerów,

T_c – czas komunikacji,

T_d – czas operacji dodatkowych zapewniających pracę równoległą (dzielenie danych, wysyłanie wartości początkowych itp.).

Przeprowadzono wstępne eksperymenty badające czas komunikacji między komputerami z wykorzystaniem RPC dla ww. metod. Przypomnieć należy, że wysyłanie komunikatu przez RPC składa się z fazy połączenia się z serwerem (otwieranie kanału), fazy wysyłania komunikatu i fazy zerwania połączenia. W przypadku kilkakrotnej komunikacji pierwszą i ostatnią fazę najlepiej wykonać jednokrotnie. Komunikaty można wysyłać wiele razy tym samym kanałem. Czas dotarcia komunikatu do oddalonego komputera zależy od wielu czynników, takich jak rodzaj medium transmisyjnego, szybkość sieci komputerowej itd. Istotnym dla nas warunkiem jest rozmiar komunikatu. Tabela 4 przedstawia wyniki eksperymentów

Tabela 4

Czas trwania poszczególnych faz przesyłu komunikatu za pomocą RPC

	Otwieranie kanału	20 bajtów	1 MB	Zamknięcie kanału
Wywołania blokujące	50	26	900	1
Wywołania nieblokujące	50	8	700	1

Wszystkie wartości są podane w ms.

Tabela 5

Przewidywany czas komunikacji dla poszczególnych wariantów rozwiązania

	Wariant 1	Wariant 2	Wariant 3	Wariant 4
Czas komunikacji	196 s	97 s	97 s	85 s

Tabela 5 przedstawia przewidywany czas komunikacji dla ww. wariantów rozwiązania. Podstawiając do wzoru (2) wartości z ww. tabel otrzymujemy:

wariant 1: $T_r = 563 + 350/3 + 196 + T_d = 875 + T_d$.

wariant 2: $T_r = 81 + 350/3 + 97 + T_d = 294 + T_d$

wariant 3: $T_r = 9 + 350/3 + 97 + T_d = 222 + T_d$

wariant 4: $T_r = 5 + 350/3 + 85 + T_d = 206 + T_d$

Jak wynika ze wstępnych obliczeń, można łatwo wybrać optymalną metodę wykorzystania RPC. Otrzymane wyniki (tabela 1) są nieco lepsze niż przewidywane (przy niewielkiej wartości T_d). Powody tych różnic są następujące:

- przedstawione czasy komunikacji w tabeli 3 przewyższają czasy w wykonanym zadaniu. Tabela zawiera przybliżone czasy dla większego rozmiaru komunikatu,
- równoległy algorytm rozwiązania jest szybszy od algorytmu szeregowego. W równoległym algorytmie można przewidywać i omijać wykonanie operacji mnożenia na elementach o wartościach zerowych. Algorytm szeregowy takich uproszczeń nie przewiduje, co powoduje dłuższy czas jego działania. Oznacza to, że wzór (2) jest tylko wzorem przybliżonym.

7. Wnioski

Jak wynika z wyżej przedstawionej analizy, istnieje kilka wariantów wykorzystania mechanizmów zdalnego wywołania procedur. Dobór właściwej metody zależy od użytkownika, od algorytmu rozwiązania zadania i od podatności zadania na rozpraszanie. Należy jednak pamiętać, że używanie najszybszej metody komunikacji między komputerami, celem przyspieszenia wykonania zadania, nie zawsze jest najbezpieczniejszym rozwiązaniem. Decyzję o wyborze między najszybszą a najpewniejszą metodą wykorzystania RPC podjąć musi użytkownik. Istnieje również możliwość przewidywania czasu równoległego rozwiązania.

LITERATURA

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

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

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

[4] Ortega J. M.: Introduction to parallel and vector solution of linear systems. Plenum Press, New York, 1988.

[5] Network Programming, Sun Microsystems.

[6] Weiss Z., Grużlewski T.: Programowanie współbieżne i rozproszone. Wydawnictwa 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.

[9] Zghidi H., Burlikowski W.: Równoległe wyznaczenie wartości własnych macierzy z wykorzystaniem mechanizmów zdalnego wywołania procedur. ZN Poł. Śl. s. Informatyka z. 31, Gliwice 1996.

Recenzent: Dr inż. Zdzisław Szczerbiński

Wpłynęło do Redakcji 29 listopada 1996 r.

Abstract

Remote Procedure Call (RPC) mechanisms can be used in networks for parallel computing. They enable users to generate distributed programs. Using these mechanisms, we can accelerate programs running time. The gain of time depends on RPC and synchronisation used mechanisms, so some of then are presented (fig. 1-5).

As example, we solve big eigensystems (matrix size 600x600) on 3 Sun Workstations in Unix network. As shown in table 1, the solution time range is very wide. So the efficiency of each presented method is discussed. We study the advantage and disadvantage of each one.

While generating parallel programs, different synchronisation mechanisms are used. Every solution variant presented in this paper is reliated with different synchronisation mechanism. For deeper analysis, we study the RPC communication and synchronisation time (tables 2-5, fig. 6-8). The presented analysis and studies allows users to approximate the final solution time. Equation 2 presents a simple model of expected solution time. This allows users to choose the best RPC using method.