ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1995

Seria: INFORMATYKA z. 29 Nr kol. 1306

Hafedh ZGHIDI

WYKORZYSTANIE MECHANIZMÓW ZDALNEGO WYWOŁANIA PROCEDUR I PAMIĘCI DZIELONEJ DO OBLICZEŃ ROZPROSZONYCH W ŚRODOWISKU SIECIOWYM Z SYSTEMEM OPERACYJNYM UNIX

Streszczenie. Niniejsze opracowanie przedstawia przykład zastosowania mechanizmów zdalnego wywołania procedur (RPC) i pamięci dzielonej do równoległych obliczeń w systemie sieciowym w środowisku systemu Unix. Przedstawia również wyniki obliczeń otrzymanych po rozwiązaniu wybranego zadania z wykorzystaniem różnej liczby komputerów.

USING REMOTE PROCEDURE CALL MECHANISMS AND SHARED MEMORY FOR DISTRIBUTED COMPUTING IN UNIX NETWORK

Summary. The paper presents an example of how to use the remote procedure call (RPC) mechanisms and shared memory for distributed computing in Unix network. It also presents the results of choosen job using different number of computers.

L'UTULISATION DES MÉCANISMES D'EXÉCUTION À LOIN ET DE LA MÉMOIRE PARTAGEÉ DANS LES RÉSAUX LOCAUX SOUS SYSTÈME UNIX POUR LES CALCULS DISTRIBUÉS

Resumé. L'article présente un example d'utulisation des mécanismes d'exécution des procédures à loin et de la mémoire partageé dans les résaux locaux sous système Unix pour calculs distribués. Contient aussi les résultats obtenus pendant la résolution d'un exercise choisi en utulisant de différents nombre d'ordinateurs.

1. Wstęp

Równoległe przetwarzanie danych może być realizowane z wykorzystaniem mechanizmów zdalnego wywołania procedur (RPC). Ze względu na ograniczenia związane z tymi mechanizmami konieczne jest stosowanie dodatkowych narzędzi. Niniejsze opracowanie przedstawia sposób wykorzystania pamięci dzielonej i mechanizmy RPC dla obliczeń rozproszonych.

Idea RPC polega na wykorzystaniu odległych komputerów do realizacji pewnych obliczeń. Mechanizmy te znalazły zastosowanie w systemach sieciowych z wykorzystaniem 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. Jednym z możliwych rozwiązań jest wykorzystanie pamięci dzielonej.

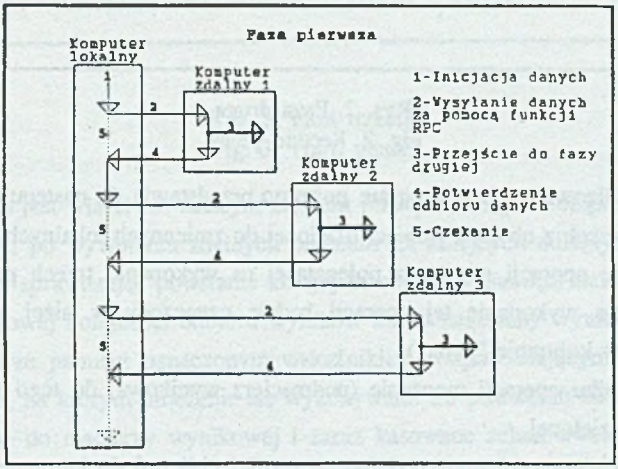
Dla zilustrowania takiego podejścia w pracy rozpatrzono problem równoległego mnożenia dwóch macierzy kwadratowych o rozmiarze n x n. Rozmiar macierzy oraz liczba

wykorzystywanych komputerów jest zmienna i ma być przekazywana jako parametr wywoływanego programu. Zadanie to realizowano w sieciowym systemie komputerowym, pracującym w systemie operacyjnym Unix i uruchamiano na stacjach roboczych Sun. Wszystkie programy zostały napisane w języku C.

2. Idea rozwiązania zadania

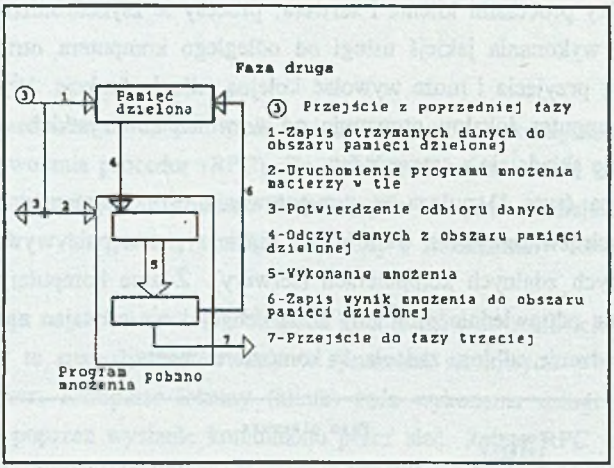
Do rozwiązania tego zadania użyto, poza RPC, pamięć dzieloną do komunikacji między procesami. Ponieważ głównym problemem przy korzystaniu z RPC była zależność czasowa pomiędzy procesami klienta i serwera, procesy te zsynchronizowano. Komputer lokalny żądając wykonania jakiejś usługi od odległego komputera otrzymuje od niego potwierdzenie jej przyjęcia i może wywołać kolejną odległą funkcję. Wyniki wywoływanych funkcji komputer lokalny otrzymuje po wywołaniu wszystkich odległych zadań. Takie rozwiązanie składa się z czterech faz.

Faza pierwsza (rys. 1) polega na przygotowaniu przez komputer klienta (program mnozrow) danych związanych z wartościami macierzy, następnie wywołaniu odległych zadań na kolejnych zdalnych komputerach (serwery). Zdalne komputery po otrzymaniu żądania wywołują odpowiednie programy (faza druga) i nie czekając na ich zakończenie wysyłają potwierdzenie odbioru zadania do komputera klienta.



Rys. 1. Faza pierwsza
Fig. 1. First phase

Fazę drugą (rys. 2) wykonuje się na komputerach odległych. Komputery te, po otrzymaniu komunikatu żądania wykonania usługi (program **mnoz_svc**), zapisują otrzymane wartości macierzy do obszaru pamięci dzielonej, a następnie wywołują program mnożenia macierzy **pobmno** w tle. Oznacza to, że nie czekają one na zakończenie jego wykonania. Następnie wysyłają potwierdzenie odbioru danych do komputera klienta, który wywołuje następną zdalną procedurę na kolejnym odległym komputerze. Program mnożenia **pobmno** wczytuje wartości macierzy z tworzonego obszaru pamięci dzielonej, wykonuje procedurę mnożenia, zapisuje wyniki do tego samego obszaru pamięci dzielonej i wywołuje program **wysyłka**, którego zadaniem jest wysłanie wynikowych danych do komputera klienta (faza trzecia).



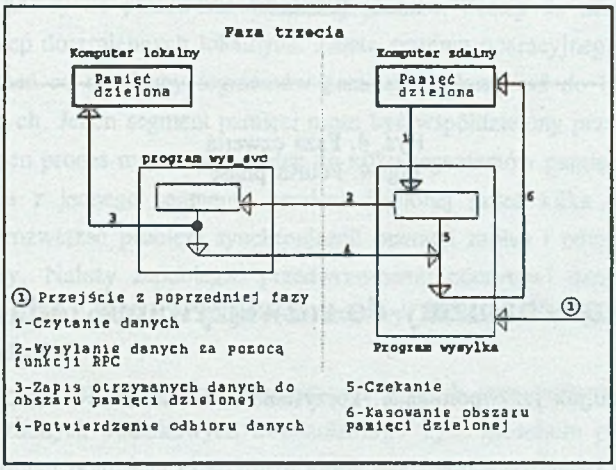
Rys. 2. Faza druga
Fig. 2. Second phase

Algorytm realizowany przez program **pobmno** przedstawia się następująco:

— odczyt danych z obszaru pamięci dzielonej do zmiennych lokalnych (listy),
— wykonanie operacji mnożenia polegającej na wykonaniu trzech pętli zagnieżdżonych (czas wykonanie tej operacji będzie oznaczony w niżej przedstawionych tabelach w kolumnie "pętla"),
— zapis wyniku operacji mnożenie (podmacierz wynikowa) do tego samego obszaru pamięci dzielonej,
— uruchomienie programu wysyłającego wyniki obliczeń do komputera klienta (program **wysyłka**),
— zakończenie działania.

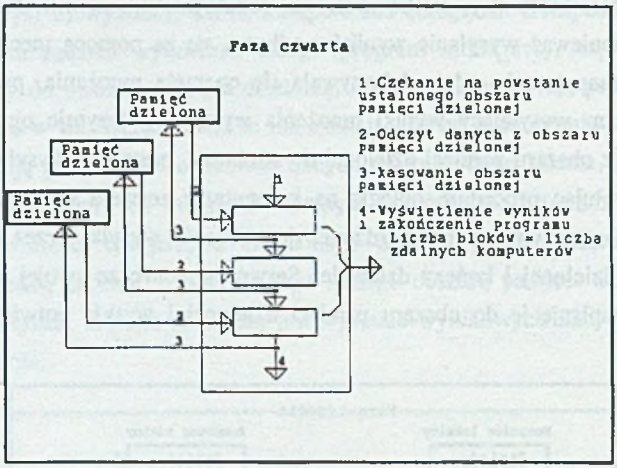
Faza trzecia (rys. 3) jest bardzo podobna do pierwszej. Polega na wysłaniu wyników mnożenia do komputera klienta, który staje się komputerem odległym (serwerem), przez

komputery odległe, które stają się komputerami lokalnymi (klientami). Rola komputerów odwróciła się, ponieważ wysyłanie wyników odbywa się za pomocą mechanizmów RPC. Na odległych komputerach, gdzie dokonywała się operacja mnożenia, program **pobmno** wywołuje program wysyłający wyniki mnożenia **wysylka**. Wczytuje on wartości wynikowej macierzy z obszaru pamięci dzielonej do struktury, następnie wysyła je do zdalnego komputera wywołując procedurę odległą na komputerze serwera (program **wys_svc**) za pomocą RPC. Po otrzymaniu potwierdzenia dostarczenia wysyłki przez serwera, kasuje obszar pamięci dzielonej i kończy działanie. Serwer wys_svc ze swojej strony po otrzymaniu danych zapisuje je do obszaru pamięci dzielonej i wysyła potwierdzenie odbioru danych.



Rys. 3. Faza trzecia
Fig. 3. Third phase

Faza czwarta jest właściwie dalszym ciągiem fazy pierwszej. Komputer lokalny (program **mnozrow**), po wywołaniu zdalnych procedur na kolejnych odległych komputerach, czeka na wyniki sprawdzając powstanie kolejnych obszarów pamięci dzielonej. Dla zapewnienia prawidłowej kolejności odbioru wyników każdy nadesłany wynik jest zapisywany w innym obszarze pamięci oznaczonym wskaźnikiem, odpowiadającym numerowi zdalnego komputera, na którym mnożenie się wykonywało. Po powstaniu obszarów danych są one wczytywane do macierzy wynikowej i zaraz kasowane celem zwolnienia zajętej pamięci operacyjnej. Po wczytywaniu wszystkich danych program **mnozrow** kończy działanie.



Rys. 4. Faza czwarta
Fig. 4. Fourth phase

3. Użyte mechanizmy do rozwiązywania zadania

W rozwiązaniu,jak już wpomniano, korzystano z mechanizmów RPC i pamięci dzielonej.

3.1. Mechanizmy RPC [1][2]

Mechanizmy zdalnego wywołania procedur użyto celem zapewnienia przesyłania danych pomiędzy komputerami. Wartości macierzy są wysyłane w strukturze o określonym formacie, zawierającej liczbę wierszy i liczbę kolumn macierzy A i podmacierzy B oraz wartości ich elementów. Ponieważ komputery pełnią rolę serwera lub klienta, zależnie od kierunku wysyłania danych, powstały dwa różne serwery, które wykonywały to samo zadanie. Różnica między nimi była tylko w formacie i typie wysyłanej struktury i danych. Pierwszy serwer **mnoz_svc** został uruchomiony na odległych komputerach. Drugi zaś **wys_svc** na komputerze lokalnym. Podczas ich generowania wraz z pozostałymi interfejsami (jak filtry, szkielety programów klientów i serwerów) użyto kompilatora **rpcgen**, a jako protokół transmisji wybrano TCP/IP, który zapewnia wysyłanie większych pakietów danych. Należy jednak podkreślić konieczność wykorzystania dwóch serwerów ze względu na różnorodność wysyłanych danych.

3.2. Pamięć dzielona [3]

Pamięć dzieloną wykorzystano w celu przekazywania danych pomiędzy różnymi niezależnymi procesami uruchamianymi na tym samym komputerze. Polega to na tworzeniu obszaru pamięci poza przestrzeniami adresowymi wszystkich działających procesów. Każdy proces pragnący uzyskać dostęp do danego segmentu wywołuje odpowiednie funkcje systemowe, odwzorowujące ten segment na jego własną przestrzeń adresową. Operacje tworzenia, kasowania, odczytu lub zapisu do tego obszaru wykonywane są za pomocą zwykłych instrukcji maszynowych. Ponieważ segment pamięci dzielonej po jego odwzorowaniu należy do przestrzeni adresowej procesu, dostęp do niego jest tak samo szybki, jak dostęp do zmiennych lokalnych. Zaletą systemu operacyjnego Unix jest to, że nie ma ograniczeń co do liczby segmentów pamięci dzielonej ani do liczby procesów z niej korzystających. Jeden segment pamięci może być współdzielony przez dowolną liczbę procesów, a jeden proces może mieć dostęp do kilku segementów pamięci dzielonej. Podczas korzystania z jednego segmentu pamięci dzielonej przez kilka procesów równocześnie należy rozwiązać problem synchronizacji operacji zapisu i odczytu danych przez aktywne procesy. Należy zapobiegać przedwczesnemu odczytowi danych przez proces odbiorcy lub zapisowi danych przez proces nadawcy. Do tego celu można użyć semaforów lub komunikatów.

Ponieważ w naszym rozwiązaniu korzystano z różnych segmentów pamięci dzielonej, nie stosowano żadnych dodatkowych mechanizmów. Tym sposobem przekazywanie danych w fazie drugiej pomiędzy programem **mnoz_svc** (odpowiadającym za odbiór danych wysyłanych przez komputer klienta) a **pobmno** (odpowiadającym za mnożenie macierzy) odbywało się właśnie za pomocą tego mechanizmu. Tak samo korzystano z pamięci dzielonej w fazie trzeciej do przekazywania danych pomiędzy programem **pobmno** a programem **wysylka** (pełniącym rolę klienta podczas wysyłania wyników mnożenia). W fazie czwartej również, podczas wysyłania danych pomiędzy wys_svc (pełniącym rolę serwera odbierającego wyniki mnożenia) a programem **mnozrow** użyto pamięci dzielonej.

4. Równoległość obliczeń

Jak już wspomniano, naszym zadaniem jest równoległe mnożenie dwóch macierzy kwadratowych o rozmiarze n x n. Celem zapewnienia równoległości obliczeń należało podzielić jedną macierz na kilka podmacierzy i wykonać ich mnożenie z drugą macierzą

na różnych komputerach równocześnie. W celu wyjaśnienia idei rozpraszania posłużymy się następującym przykładem: Mamy do mnożenia dwie macierze macA i macB o rozmiarze 5x5 (5 wierszy i 5 kolumn), korzystając z 2 komputerów. Należy więc określić wielkość podmacierzy macierzy macB, tzn. liczby kolumn, posługując się wzorami:
liczba kolumn = 5 / 2; reszta = 5 modulo 2.
Oznacza to że podmacierz pierwsza macB0 będzie miała 2 kolumny (liczba kolumn), druga natomiast macB1 3 kolumny (liczba kolumn + reszta).

$$ilosckolumn = 5 / 2 = 2 \;;\; reszta = 5 \bmod 2 = 1.$$

$$macB = \begin{bmatrix} a11 & a12 & a13 & a14 & a15 \\ a21 & a22 & a23 & a24 & a25 \\ a31 & a32 & a33 & a34 & a35 \\ a41 & a42 & a43 & a44 & a45 \\ a51 & a52 & a53 & a54 & a55 \end{bmatrix} \Rightarrow macB0 = \begin{bmatrix} a11 & a12 \\ a21 & a22 \\ a31 & a32 \\ a41 & a42 \\ a51 & a52 \end{bmatrix} \; macB1 = \begin{bmatrix} a13 & a14 & a15 \\ a23 & a24 & a25 \\ a33 & a34 & a35 \\ a43 & a44 & a45 \\ a53 & a54 & a55 \end{bmatrix}$$

Po wysłaniu danych na odległe komputery i wykonaniu mnożenia, dostajemy w odpowiedzi jako wynik mnożenia podmacierz o wielkości równej wysyłanej, ponieważ wynik mnożenia macierzy z podmacierzą jest podmacierzą. Po otrzymaniu kolejnych podmacierzy następuje ich składanie do macierzy wynikowej macC.

Liczba wykorzystanych komputerów = 2 :

$$C = A * B = A * (B0 \square B1) = A * B0 \square A * B1$$

Liczba wykorzystanych komputerów = n :

$$C = A * B = A * (B0 \square B1 \square ... \square Bn) =$$
$$= A * B0 \square A * B1 \square ... \square A * Bn$$

Organizacja obliczeń przedstawia się następująco:

- uruchomienie serwerów odpowiadających za mnożenie macierzy na odległych komputerach (program **mnoz_svc**),
- uruchomienie serwera odpowiadającego za otrzymanie podmacierzy wynikowych na komputerze lokalnym (program **wys_svc**),
- uruchomienie programu mnożenia równoległego na komputerze lokalnym (program klienta **mnozrow**), który wykonuje:
  - inicjację wartości macierzy i podział macierzy B na odpowiednią liczbę podmacierzy,
  - wysłanie kolejnych macierzy na odległe komputery i równoległe wykonanie na nich mnożenia,
  - czekanie na powstanie kolejnych obszarów pamięci dzielonej z wartościami podmacierzy wynikowych,

— złożenie macierzy wynikowej,
— zakończenie działanie programu.

5. Wyniki obliczeń

Tę część opracowania poświęcimy omówieniu uzyskanych wyników mnożenia macierzy z wykorzystaniem wyżej opisanych mechanizmów. Pierwszy przykład ich zastosowania przedstawia niniejsza tabela.

Tabela 1

Wyniki przykładu pierwszego

Komputer	Pętla	Mnożenia	Całość	Ipx
classic2	38 sek.	40 sek.	44 sek.	77 sek.
sun10	27 sek.	28 sek.		

Tabela ta przedstawia wynik mnożenia dwóch macierzy o rozmiarze 300x300 z wykorzystaniem dwóch komputerów. Program **mnozrow** (w tym jak i w niżej przedstawionych przykładach), odpowiadający za rozsyłania zadania i wyświetlania wyników końcowych, został uruchomiony na trzecim komputerze ipx. Oznacza to, że ipx pełnił rolę lokalnego komputera (klienta) realizując główną część programu, natomiast pozostałe wykorzystane komputery były zdalnymi serwerami.

Kolumna druga, oznaczona jako "Pętla", zawiera czas wykonania tylko pętli mnożenia macierzy macA z podmacierzą macB (podmacierz macB0 lub macB1). Kolumna oznaczona jako "Mnożenia" przedstawia czas wykonania całego programu **pobmno**, natomiast kolumna "Całość" określa czas wykonania całego zadania (program **mnozrow**). Wyraźnie widać, że komputer Sun10 jest szybszy od Classic2. Operacje związane z przesyłem danych pomiędzy komputerami oraz zapisem i odczytem z pamięci dzielonej powodują opóźnienie zakończenia zadania o kilka sekund, licząc od momentu zakończenia mnożenia na wolniejszym komputerze classic2. Ostatnia kolumna przedstawia czas mnożenia wyżej wymienionych macierzy tylko na jednym komputerze (classic1), bez rozpraszania. Jak widać, korzystanie tylko z dwóch komputerów niewiele przespiesza obliczenia.
Drugi przykład ilustruje rozwiązanie tego samego zadania z wykorzystaniem dodatkowego komputera classic3.

Tabela 2

Wyniki przykładu drugiego

Komputer	Pętla	mnożenia	Całość	Ipx
classic3	26 sek.	28 sek.	34 sek.	77 sek.
classic2	26 sek.	27 sek.		
sun10	19 sek.	20 sek.		

Łatwo zauważyć, że korzystanie z większej liczby komputerów przespiesza wykonanie zadania. Udowadnia to prawidłowość założeń i algorytmu rozwiązywania. Należy również podkreślić, że na ocenę otrzymanego zysku czasowego ma wpływ nie tylko ilość wykorzystywanych komputerów, lecz również rozmiar macierzy (tabela 3). Mnożenie macierzy o rozmiarze 200x200 na jednym komputerze trwa niewiele dłużej (28 sek.) niż na dwóch (19 sek.). Jest to spowodowane kosztami komunikacji między komputerami (wysyłania danych przez sieć) oraz kosztami operacji tworzenia, zapisu i odczytu z obszarów pamięci dzielonej. Natomiast mnożenie macierzy o rozmiarze 400x400 na dwóch komputerach jest o 47 sek. szybsze niż na jednym. Oznacza to, że istnieje pewna zależność między czasem wykonania pętli w programie mnożenia **pobmno** a czasem przesyłu danych między komputerami. Ogólnie można stwierdzić, że najlepsze wyniki można otrzymać podczas mnożenia macierzy o większych rozmiarach z wykorzystaniem większej liczby komputerów. Taki wynik otrzymaliśmy przy mnożeniu macierzy o rozmiarze 400x400, korzystając z 4 komputerów. Czas wykonania tego zadania wynosił 61 sek., natomiast na jednym komputerze 91 sek. dłużej.

Tabela 3

Wpływ rozmiaru macierzy i liczby komputerów na czas mnożenia

Rozmiar macierzy	Liczba komputerów	Czas
200x200	1	28 sek.
200x200	2	19 sek.
400x400	1	152 sek.
400x400	2	105 sek.
400x400	4	61 sek.

LITERATURA

[1] Sun Microsystem: Network Programming.

[2] WEISS Z., GRUŻLEWSKI T.: Programowanie współbieżne i rozproszone, wydawnictwo Naukowo-Techniczne Warszawa, 1993.

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

Recenzent: Dr inż. Andrzej Wilk

Wpłynęło do Redakcji 17 listopada 1995 r.

Abstract

In unix network, exists many useful ways of communication between remote computers. One of them is RPC (Remote Procedure Call), which garantee fast and good communication. Using these mechamisms, we are able to run remote programs (procedures) in a klient/server model. Programmers who want to develop a distributed algorytm, can make use of these mechanisms by calling independent parts of their algorytms at the same time on different machines. Therefore, to implement distributed algorytms, additional mechanisms are needed. The paper presents how to use RPC and shared memory to implement distributed large size matrix multiplication algorytm. It present and idea, based on dividing the algorytm on four phases shown in fig. 1 to fig. 4. To demonstrate the efficiency of the used metod and algorytm we present the time profit in the function of matrix size, intercomputer communication and the number of remote computers used. As mentionned, the time profit is greater when the matrix size is larger and the number of used machines is bigger. Table 1 to 3 present the obtained results in different tests.