STUDIA INFORMATICA 2002
Volume 23 Number 3 (50)

Hafedh ZGHIDI
Politechnika Śląska, Instytut Informatyki

WYKORZYSTANIE ROZPROSZONYCH OBSZARÓW PAMIĘCI DZIELONEJ I MECHANIZMY ZDALNEGO WYWOŁANIA PROCEDUR DO PROGRAMOWANIA RÓWNOLEGŁEGO

Streszczenie. Niniejsza praca przedstawia nową metodę wykorzystania mechanizmów zdalnego wywołania procedur do obliczeń równoległych. Zaproponowane rozwiązanie wykorzystujące rozproszone obszary pamięci dzielonej pozwala na zwiększenie efektywności mechanizmów RPC. Do jej oceny przedstawimy przykład jej zastosowania do rozwiązania problemu równoległego wyszukiwania wzorca w bazie danych obrazów cyfrowych.

PARALLEL PROGRAMMING USING DISTRIBUTED SHARED MEMORY AND REMOTE PROCEDURE CALL MECHANISMS

Summary. The paper presents a new method of using Remote Procedure Call mechanisms for parallel programming. The proposed method which uses shared distributed memory increase RPC efficiency. An example of how to use the proposed method is also presented. It also allows to evaluate its purposefulness.

1. Wstęp

Rosnące zapotrzebowanie na komputery o dużej mocy obliczeniowej do rozwiązywania w ograniczonym czasie coraz bardziej skomplikowanych zagadnień wymusza konieczność ciągłego udoskonalania technologii i architektury komputerów jednoprocesorowych. Innym rozwijanym kierunkiem nauki pozwalającym na rozwiązanie tego problemu jest tworzenie systemów wieloprocesorowych łączących dużą liczbę komputerów. Na powodzenie systemów wielokomputerowych złożyły się: szybki rozwój komputerów osobistych, ich popularność, dostępność i gwałtowny rozwój sieci komputerowych. Łączenie większej liczby

tańszych komputerów i ich równoległe wykorzystanie jest dużo korzystniejsze niż posiadanie pojedynczego bardzo wydajnego komputera.

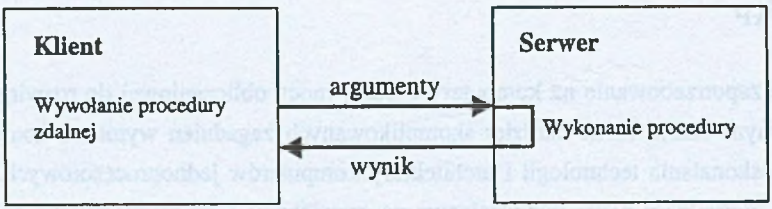
Do dostępnych narzędzi programowych pozwalających na wykorzystanie systemów wieloprocesorowych do obliczeń równoległych można zaliczyć: system PCN, system Piranha, system Paradise oraz inne systemy i narzędzia, takie jak RMS i biblioteka P4 [1,2]. Do najpopularniejszych narzędzi zaliczyć należy systemy: PVM, MPI, Linda oraz mechanizmy zdalnego wywołania procedur (RPC, ang.Remote Procedure Call) [3,4].

Niniejsza praca przedstawia nową metodę wykorzystującą obszary pamięci dzielonej i mechanizmy RPC do obliczeń równoległych. Zaproponowana metoda ma na celu zwiększenie efektywności wykorzystania RPC do obliczeń równoległych i rozproszonych. Do oceny efektywności zaproponowanej metody porównamy uzyskane wyniki z wynikami otrzymanymi przy wykorzystaniu systemów PVM i Linda i tradycyjnymi mechanizmami RPC.

2. Mechanizmy zdalnego wywołania procedur

Mechanizmy RPC [3,4,5] są wysokopoziomowymi protokołami komunikacyjnymi, pozwalającymi na tworzenie sieciowych aplikacji rozproszonych według modelu klient/serwer. Mechanizmy RPC poprzez specjalnie przygotowane procedury zasłaniają niskopoziomowe sieciowe mechanizmy, ułatwiając tym samym zadanie programistom.

Idea działania RPC jest następująca: aplikacja wykonywana przez komputer A (klient) wywołuje procedurę znajdującą się na komputerze B (serwer), wysyłając do tego ostatniego polecenie wykonania podprogramu wraz z argumentami wywołania. Następnie program wywołujący przechodzi w stan oczekiwania na zakończenie wykonania podprogramu przez komputer B, aby odebrać zwracane przez ten podprogram wyniki. Rysunek 1 przedstawia zasadę działania RPC.



Rys. 1. Schemat działania RPC
Fig. 1. RPC functioning scheme

Taki model wykorzystania RPC (asynchroniczny) nie pozwala na ich wykorzystanie do obliczeń równoległych. Istnieje natomiast wiele wariantów zastosowania mechanizmów RPC w trybie synchronicznym (nieblokującym). W pracach [6,7,8] przedstawiono kilka metod zdalnego wywołania procedur umożliwiających równoległe wykonanie programów. W większości przypadków wykorzystano dwa serwery RPC (działające w trybie nieblokującycm). Jeden był uruchomiony na zdalnych komputerach (serwerach) i umożliwił wykorzystanie ich mocy obliczeniowej do realizacji części obliczeń. Drugi uruchomiono na komputerze sterującym (klienta), jego zadaniem było zrealizowanie wszystkich etapów programu równoległego (odbiór wyników pośrednich, synchronizacja obliczeń itd.).

W celu zrównoważenia obciążenia wykorzystanych zdalnie komputerów zastosowano dynamiczny przydział zadań [5,8]. Metoda ta polegała na informowaniu komputera klienta o zakończeniu obliczeń i gotowości komputera serwera do realizacji kolejnego zadania. Dynamiczny podział zadań (w przeciwieństwie do statycznego podziału zadań) pozwala na lepsze wykorzystanie mocy obliczeniowej komputerów, co może przyczynić się do uzyskania dużych korzyści czasowych [5,8]. Metoda ta ma jednak następujące ograniczenia:

- warunkiem jej wykorzystania jest całkowita niezależność pomiędzy wykonywanymi zadaniami,
- wykorzystanie jej wiąże się zwykle z koniecznością zastosowania dodatkowych mechanizmów synchronizacji (semafory, sygnały itd.) oraz przesyłanie dodatkowych danych umożliwiających identyfikację komputera zgłaszającego gotowość do wykonania kolejnego zadania,
- zastosowanie tej metody powoduje wzrost liczby operacji wymiany danych pomiędzy komputerem sterującym a komputerami zdalnymi, co powoduje wzrost obciążenia sieci,
- komputery zdalne od momentu zgłoszenia gotowości do realizacji kolejnego zadania do momentu jej otrzymania są bezczynne, uzależnia to program serwera od programu klienta,
- metoda ta jest opłacalna tylko dla czasochłonnych lub licznych zadań.

W celu minimalizacji czasu realizacji programu równoległego, przy zastosowaniu dynamicznego przydziału zadań, należy maksymalnie uniezależnić program klienta od programu serwera i zminimalizować ilość operacji wymiany danych i synchronizacji pomiędzy komputerami. W tym celu zastosowano rozwiązanie, przypominające ideę działania systemu Linda [1,2,5], nazywane dalej RPCL.

Efektywność zaproponowanej metody RPCL będziemy testować wykorzystując jako zadanie problem wyszukiwania w obrazie cyfrowym fragmentu identycznego z zadanym wzorcem. Algorytm ten przypomina przeszukiwanie wzorca literowego w tekście [5]. Do

dyspozycji mamy obraz I_0 o rozmiarach n_0 x m_0 oraz wzorzec I_1 o rozmiarach n_1 x m_1. Rozmiary obrazów spełniają warunek:

$$n_0 \geq n_1, m_0 \geq m_1 \quad (1)$$

Przeszukiwanie wzorca polega na sekwencyjnym przeglądaniu kolejnych pikseli p, $p \in I_0, I_1$ o współrzędnych (i, j) i porównywaniu ich wartości (v_{p0}, v_{p1}).Współrzędne (i, j) należą do podmacierzy Z o rozmiarach $(n_0 - n_1)x(m_0 - m_1)$. Zapewnia to poszukiwanie wzorca I_1 w obrębie obrazu I_0. Jeśli wartości v_{p0}, v_{p1} są różne, to dla współrzędnych (i, j) obraz i wzorzec są różne. Jeśli dla wszystkich pikseli wzorca $v_{p1} = v_{p0}$, to dla współrzędnych (i, j) obraz i wzorzec są identyczne.

Wyżej wymieniony algorytm charakteryzuje się małą złożonością obliczeniową, jego zrównoleglenie nie przyniesie dużych korzyści czasowych i może być bezcelowe. Straty wynikające z rozpraszania obliczeń, tj. uruchomienie procedur zdalnych, przesył komunikatów do komputerów oraz synchronizacja obliczeń, mogą przewyższać (szczególnie dla małych obrazów) zysk czasowy wynikający ze zrównoleglenia obliczeń. Zysk czasowy można natomiast uzyskać w przypadku, gdy obliczeń jest dużo więcej, np. podczas przeszukiwania wzorca w bazie danych obrazów.

Zakładając, że do dyspozycji mamy bazę danych obrazów, a do wyszukiwania jest jeden wzorzec, kolejne rekordy bazy danych (a ściślej, obrazy dostępne poprzez adresy zawarte w rekordach) są równocześnie porównywane ze wzorcem na różnych serwerach obliczeniowych. Każdy serwer wykonuje porównywanie wzorca z innym rekordem bazy danych (obrazem). Wykorzystuje się przy tym sekwencyjny program porównywania obrazów. W przypadku gdy do porównywania jest więcej wzorców, następuje powtórzenie obliczeń dla każdego wzorca. Opracowanie programu równoległego dla dużej bazy danych może być opłacalne. W tym celu została utworzona baza danych "pictures" w sieci stacji roboczych *Sun* z wykorzystaniem serwera relacyjnej bazy danych Ingres 6.4 [9,10]. Opracowana baza danych "pictures" składa się z jednej tabeli "picts", której elementami są nazwy plików (obrazów), ich parametry (rozmiary) oraz ścieżka ich umiejscowienia na nośniku zewnętrznym komputera (dysku).

Tak przedstawiony problem wyszukiwania wzorca w bazie danych obrazów charakteryzuje się dużą niezależnością pomiędzy kolejnymi wykonywanymi zadaniami. Stanowi więc dobry przykład wykorzystania dynamicznego przydziału zadań w nowym modelu rozwiązania RPCL.

3. Opis metody RPCL

Metoda ta polega na utworzeniu na zdalnych komputerach przestrzeni komunikacyjnej (PK), składającej się z określonej liczby obszarów pamięci dzielonej (obszary wymiany OW), przypominającej przestrzeń krotek w systemie Linda. Liczba OW (rozmiar PK) jest podana przez użytkownika jako parametr programu. Programy klienta i serwera działają, podobnie jak w Lindzie, na zasadzie nadzorca/wykonawca. Program klienta generuje nowe zadania, program serwera je wykonuje.

W porównaniu do przestrzeni krotek systemu Linda, przestrzeń komunikacyjna różni się tym, że:

- każdy obszar wymiany (OW) stanowi odrębny obszar pamięci dzielonej o określonej strukturze (rysunek 2),
- jest to rozproszona, ale nie współdzielona przestrzeń komunikacyjna. Dostęp do przestrzeni komunikacyjnej (PK) mają tylko procesy działające na komputerze, na którym przestrzeń została stworzona,
- wszystkie OW są tego samego formatu - nie ma możliwości doboru OW poprzez dopasowanie wzorca,
- dostęp do określonego OW jest możliwy jedynie poprzez jego adres (adres obszaru pamięci dzielonej),
- liczba obszarów wymiany OW (*MaxJob*) jest ustalona podczas tworzenia PK (podana przez użytkownika) i może być zależna od ograniczeń systemowych (dostępnej pamięci komputera, jednostki zarządzania pamięcią MMU, konfiguracji komputera itp.),
- możliwy jest tylko odczyt lub zapis do istniejącego OW, bez możliwości dodawania czy usuwania OW z przestrzeni komunikacyjnej,

JobNr	Width	Height	PicName
Nr zadania	*Szerokość obrazu*	*Wysokość obrazu*	*Nazwa obrazu*

Rys. 2. Struktura obszaru wymiany (OW)
Fig. 2. Exchange area structure

Pole *PicName* wskazuje pełną nazwę obrazu (nazwa zbioru graficznego wraz ze ścieżką jego umiejscowienia na dysku komputera). W celu uproszczenia struktury przestrzeni komunikacyjnej nie zawiera ona obszarów wymiany z opisem wzorca. Parametry wyszukiwanego wzorca podaje użytkownik jako parametr wejściowy programu.

Tak zorganizowana przestrzeń komunikacyjna pełni rolę bufora, który umożliwia dostęp programu klienta i serwera do zadań. Program klienta (nadzorca) dopisuje kolejne zadanie do wykonania do **PK** istniejącego **OW**. Program serwera (wykonawca) pobiera zadanie z istniejącego w **PK** **OW** i wykonuje obliczenia. Obie operacje (zapis do i odczyt z **PK**) mogą się wykonywać równocześnie, ale każda dla innego **OW**.

W celu zapobiegania zapisywaniu zadań do obszaru wymiany **OW**, który nie został jeszcze odczytany, można zastosować semafor niebinarny (*SemJob*), pełniący rolę licznika zadań. Program klienta na początku działania tworzy na każdym zdalnym komputerze semafor *SemJob* i przestrzeń komunikacyjną **PK**. Następnie zapełnia każdą **PK**, zapisując na każdym zdalnym komputerze tyle **OW**, ile wynosi wartość parametru *MaxJob* i ustawia odpowiednio wartość *SemJob*. Wartość każdego semafora jest więc w początkowej fazie działania programu równa rozmiarowi **PK** (*SemJob=MaxJob*). Następnie nadzorca przechodzi w fazę zawieszenia, wywołując funkcję *sleep()*, co pozwala komputerom zdalnym na realizację pewnej liczby zadań. Program serwera po odczytaniu wartości **OW** zmniejsza wartość semafora (*SemJob=SemJob-1*). Program klienta, przed umieszczeniem kolejnych nowych zadań w **PK**, sprawdza liczbę wykonanych zadań (liczba odczytanych **OW**) (*DoneJob=MaxJob-SemJob*), następnie zapisuje do **PK** tyle zadań, ile wynosi *DoneJob* i zwiększa wartość semafora (*SemJob=MaxJob*). Jeśli liczba pozostałych do wykonania zadań jest mniejsza od *DoneJob*, wartość semafora jest zwiększana o odpowiednią wartość. W celu zakończenia pracy komputerów zdalnych i usunięcia **PK** program klienta zapisuje do **OW** zadanie sygnalizujące zakończenie obliczeń, analogia do tzw. "trującej pigułki", charakteryzującej się ujemnym numerem zadania (*JobNr* = -1).

4. Zastosowanie metody RPCL do problemu wyszukiwania wzorca w bazie danych obrazów

Do rozwiązania ww. problemu, przy zastosowaniu metody RPCL, przyjęto następujące założenia:

1. liczba **OW** (rozmiar **PK**) wynosi 5,
2. do dyspozycji jest 5 komputerów zdalnych (*Classic1, Classic2, Classic3, Sun10 i Ipx*), pełniących rolę serwerów obliczeniowych (wykonawców),
3. komputerem sterującym jest *Sun10*, który jest równocześnie serwerem bazy danych,
4. do wyszukiwania jest jeden wzorzec, a liczba rekordów bazy danych jest dużo większa od liczby komputerów pomnożonej przez liczbę obszarów wymiany.

Rolą komputera sterującego jest połączenie się z bazą danych, pobranie kolejnych jej rekordów i umieszczenie w **PK** komputerów zdalnych zadań do wykonania. Opisy tych zadań w postaci: numer zadania i parametry obrazu tworzą nową zawartość obszarów wymiany.

Celem opracowanej metody RPCL jest eliminacja dwóch następujących faz programu równoległego:

1. W fazie pierwszej komputer zdalny zamienia się rolami z komputerem lokalnym (komputer zdalny staje się klientem, a komputer sterujący staje się serwerem) w celu przesłania swojego identyfikatora oraz zgłoszenia zakończenia obliczeń i gotowości do realizacji kolejnego zadania.
2. W fazie drugiej komputer sterujący, po identyfikacji komputera zdalnego gotowego do wykonania obliczeń, przesyła mu kolejne zadanie.

Eliminacja tych dwóch faz pozwala na zmniejszenie czasu wykonania programu równoległego i bardziej efektywne wykorzystanie komputerów zdalnych. Do realizacji tego modelu wykorzystano dwa serwery RPC. Oba serwery zostały uruchomione na komputerach zdalnych.

Zadaniem pierwszego serwera RPC jest wykonanie obliczeń związanych z porównywaniem obrazów. Polega to na pobraniu **OW** z **PK**, dekrementacji wartości semafora *SemJob* i wykonaniu procedury porównywania wzorca z obrazem.

Jak już wspomniano, dostęp do **PK** mają tylko procesy działające na komputerze, na którym przestrzeń została utworzona. Dlatego rolą drugiego serwera RPC jest umożliwienie komputerowi sterującemu (a dokładnie programowi klienta) dostępu do przestrzeni komunikacyjnej tworzonej na zdalnych komputerach w celu zapisywania kolejnych zadań do **PK**. Program klienta (nadzorca) po rozpoczęciu działania wypełnia całą przestrzeń komunikacyjną na wszystkich zdalnych komputerach, ustawia wartość semafora *SemJob* i uruchamia procedurę porównywania wzorca z obrazem. Następnie przechodzi do fazy chwilowego uśpienia, wywołując funkcję *sleep()*, co pozwala zdalnym komputerom na wykonanie części zadań.

W celu dopisywania zadań do **PK** komputer sterujący sprawdza wartość semafora (*SemJob*) na każdym zdalnym komputerze i zapisuje do **PK** odpowiednią liczbę zadań (*DoneJob* = *MaxJob* - *SemJob*). Jeśli liczba pozostałych rekordów bazy danych jest mniejsza od *DoneJob*, zapisywanych jest tyle zadań, ile zostało rekordów, a wartość semafora jest zwiększona o odpowiednią wartość. Po wyczerpaniu wszystkich rekordów bazy danych program klienta umieszcza w **PK** komputerów zdalnych "trującą pigułkę" w celu zakończenia działania procedur zdalnych.

Dostęp do **PK** w postaci zapisu i odbioru **OW** jest cykliczny. Program serwera (wykonawca) pobiera kolejne **OW** poczynając od pierwszego. Po pobraniu ostatniego,

pobiera ponownie pierwszy. Program klienta po sprawdzeniu liczby wykonanych zadań zapisuje nowe zadania do pobranych **OW**. Obszary wymiany są więc przeznaczone do wielokrotnego wykorzystania.

W tym modelu poza organizacją obliczeń należy brać pod uwagę konieczność rozwiązania wielu problemów, mogących powodować nieefektywne wykorzystanie komputerów lub niepoprawne wykonanie programu równoległego:

a) należy dbać o to, żeby komputery zdalne zawsze miały zadania do wykonania. Pozwala to szybciej zakończyć wykonanie programu równoległego,
b) należy zapobiegać pobraniu **OW** z tym samym zadaniem,
c) po pobraniu "trującej pigułki" należy zakończyć działanie programu serwera dopiero po wykonaniu wszystkich zadań.

Pierwszy problem można rozwiązać zwiększając rozmiar przestrzeni komunikacji i zapewniając częste sprawdzanie przez komputer sterujący liczby zdalnie wykonanych zadań (liczby pobranych **OW**). Można to realizować zmniejszając parametr funkcji *sleep()*.

Drugi problem rozwiązano wprowadzając numerację zadań (pole *JobNr*). Warunkiem wykonania zadania jest to, że numer pobranego zadania jest większy od numeru ostatnio wykonywanego.

Trzeci problem rozwiązano w ten sposób, że po pobraniu "trującej pigułki" program serwera zakończy działanie dopiero po sprawdzeniu, czy numery zadań we wszystkich **OW** nie są większe od numeru ostatnio wykonywanego zadania.

Program wyszukiwania wzorca w bazie danych obrazów przy wykorzystaniu metody RPCL składa się z następujących etapów:

Komputer klienta:

K1: inicjacja komputerów zdalnych,

K2: uruchomienie we wszystkich zdalnych komputerach procedury tworzenia przestrzeni komunikacyjnej (obszarów pamięci dzielonej) i semafora niebinarnego *SemJob*. W tym etapie wykorzystane są wywołania blokujące RPC, gdyż program klienta (komputer sterujący) musi otrzymać adresy obszarów pamięci dzielonej i identyfikatory semaforów,

K3: połączenie się z bazą danych i wczytanie rekordów tabeli "picts" (do przeglądania zawartości bazy danych wykorzystano mechanizm kursora),

K4: umieszczenie w **PK** zadania i zwiększenie wartość semafora (liczba zadań jest równa rozmiarowi **PK**: *SemJob=MaxJob*),

K5: wysłanie do zdalnych komputerów parametrów wzorca,

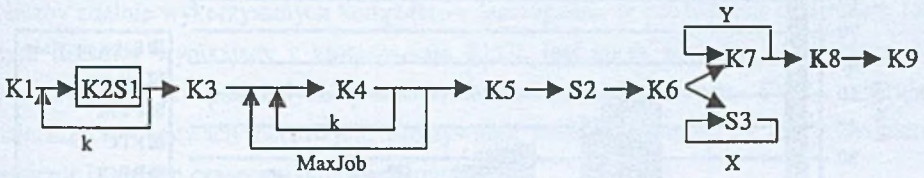
K6: wywołanie na zdalnych komputerach procedury porównania wzorca z obrazem,

K7: sprawdzenie wartości semafora *SemJob*, umieszczanie w **PK** odpowiedniej liczby nowych zadań i zwiększenie wartości semafora,

K8: umieszczenie w **PK** "trującej pigułki",
K9: odbiór wyników porównywania wzorca z obrazami i zakończenie programu.

Komputer serwera:
S1: utworzenie przestrzeni krotek i semafora oraz wysyłanie do lokalnego komputera adresów obszarów pamięci dzielonej (**OW**) i identyfikatora semafora (*SemJob*),
S2: wczytanie wzorca,
S3: pobranie z **PK** kolejnego **OW**, zmniejszanie wartości semafora, wykonywanie procedury porównywania obrazu ze wzorcem i zapis wyniku obliczeń,
S4: zakończenie programu po pobraniu "trującej pigułki".

Jak już wspomniano, do rozwiązania zadania wykorzystano 2 serwery RPC. Etap K1 jest wykonywany za pomocą obu serwerów. Etapy K2, K5 i K6 są wykonywane za pomocą pierwszego serwera. Etapy K4, K7 i K8 są wykonywane za pomocą drugiego serwera. Etapy K3 i K9 są wykonywane na komputerze lokalnym. Rysunek 3 przedstawia przebieg czasowy kolejnych etapów rozwiązania.



Operacja blokująca
k - liczba wykorzystanych komputerów
X - do pobrania trującej pigułki
Y - do wyczerpania rekordów bazy danych

Rys. 3. Etapy rozwiązania równoległego
Fig. 3. Parallel program phases

5. Wyniki badań

Celem przeprowadzonego eksperymentu było porównanie czasu działania programu wykorzystującego metodę RPCL z innymi rozwiązaniami wykorzystującymi systemy Linda, PVM i klasyczny RPC. Pozwoli to na ocenę zaproponowanej metody nie tylko względem rozwiązania sekwencyjnego, ale również względem innych narzędzi programowych.
Tabela 1, rysunki 4 i 5 przedstawiają wyniki pomiarów.

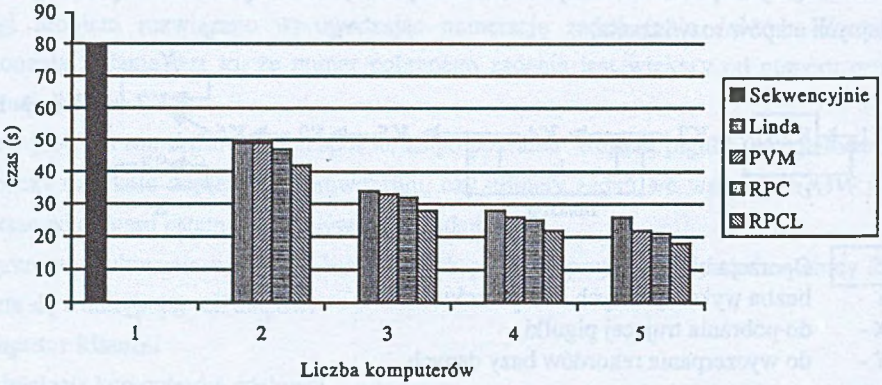
Analizując otrzymane wyniki, można stwierdzić, że wszystkie wykorzystane narzędzia programowe pozwalają na uzyskanie lepszych wyników czasowych niż w przypadku

rozwiązania sekwencyjnego. Najlepsze wyniki otrzymuje się jednak dla programu wykonywanego za pomocą mechanizmów RPC i zaproponowanej metody RPCL.

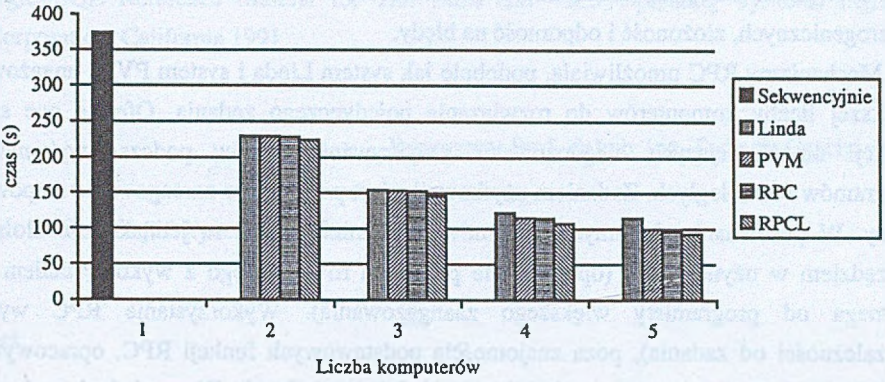
Tabela 1

Czas (w [s]) wykonania programu w systemach Linda, PVM, RPC, RPCL

Liczba rekordów	100				500			
Liczba komputerów	2	3	4	5	2	3	4	5
Linda	49	34	28	26	230	155	123	116
PVM	49	33	26	22	230	154	116	100
RPC	47	32	25	21	229	153	115	98
RPCL	42	28	22	18	225	146	108	93
Rozwiązanie sekwencyjne	79				375			



Rys. 4. Czas wykonania programu dla 100 rekordów BD
Fig. 4. Program elapsed time for 100 records



Rys. 5. Czas wykonania programu dla 500 rekordów BD
Fig. 5. Program elapsed time for 500 records

Czasy realizacji programu równoległego przy wykorzystaniu RPC i RPCL są w przedstawionych badaniach najmniejsze. Wraz ze wzrostem liczby rekordów bazy danych i liczby zdalnie wykorzystanych komputerów (szczególnie w porównaniu do systemu Linda) zysk czasowy wynikający z zastosowania RPCL jest coraz większy. Mechanizmy RPC, w porównaniu do pozostałych prezentowanych systemach (Linda, PVM), operują na najniższych warstwach sieciowych. Efektywność transmisji danych za ich pośrednictwem owocuje krótszym czasem wykonania programów.

6. Podsumowanie

Zaletą nowego modelu jest współbieżne działanie programu klienta i serwera, zmniejszenie liczby i czasu wykonania operacji synchronizacji i transmisji danych oraz wyeliminowanie fazy oczekiwania programu serwera na otrzymanie kolejnego zadania do wykonania. Dzięki temu nowa metoda zapewnia uzyskanie najlepszych wyników czasowych. Stworzenie modelu wzorowanego na systemie Linda, wykorzystującego mechanizmy RPC, przyczyniło się do efektywnego wykorzystania komputerów zdalnych. Choć tworzona przestrzeń komunikacyjna i operacje na niej wykonywane są ograniczone (w porównaniu do mechanizmów Lindy), to udało się uzyskać lepsze wyniki czasowe, co dowodzi, że mechanizmy RPC charakteryzują się dużą efektywnością czasową.

Oceniając efektywność przedstawionych narzędzi i systemów, opieraliśmy się wyłącznie na porównaniu czasu wykonania za ich pomocą programu równoległego. Jest to jedna, ale nie jedyna miara wykorzystana do tego celu. Należy tu uwzględnić jeszcze inne aspekty, takie

jak: dostępność narzędzi, skalowalność, przenośność, możliwość działania w środowiskach heterogenicznych, złożoność i odporność na błędy.

Mechanizmy RPC umożliwiają, podobnie jak system Linda i system PVM, angażowanie większej liczby komputerów do rozwiązania pojedynczego zadania. Oferują one szereg funkcji umożliwiających diagnozowanie i wykrywanie błędów podczas wykonywania programów równoległych. Zadaniem użytkownika jest prawidłowe zareagowanie na powstałe błędy. W porównaniu do innych systemów mechanizmy RPC są jednak dość złożonym narzędziem w użytkowaniu (opracowanie programu równoległego z wykorzystaniem RPC wymaga od programisty większego zaangażowania). Wykorzystanie RPC wymaga (w zależności od zadania), poza znajomością podstawowych funkcji RPC, opracowywania dodatkowych mechanizmów synchronizujących lub kontrolnych. Biorąc jednak pod uwagę, że głównym celem rozpatrywanym w tej pracy jest zmniejszenie czasu wykonania zadań, można stwierdzić, na podstawie wykonanych badań, że mechanizmy RPC są jednym z najlepszych narzędzi.

LITERATURA

1. Zieliński K. -red: Środowiska programowania rozproszonego w sieciach komputerowych, Kraków 1994.
2. Kozielski S. -red: Systemy umożliwiające realizację algorytmów równoległych w sieciach komputerowych. Politechnika Śląska, Skrypty uczelniane Nr 1975, Gliwice 1996.
3. Network Programming Guide. Sun Microsystems 1990.
4. Gabassi M., Dupouy B.: Przetwarzanie rozproszone w systemie Unix. LUPUS, Warszawa 1995.
5. Zghidi H.: Efektywność obliczeń równoległych w sieci komputerowej przy wykorzystaniu mechanizmów zdalnego wywołania procedur. Rozprawa doktorska. Politechnika Śląska, Gliwice 2000.
6. Gabassi M., Dupouy B.: Przetwarzanie rozproszone w systemie Unix. LUPUS, Warszawa 1995.
7. Stevens W. Richard: Programowanie zastosowań sieciowych w systemie Unix. Wydawnictwo Naukowo-Techniczne, Warszawa 1995.
8. Zghidi H.: Wpływ metody wykorzystania mechanizmów zdalnego wywołania procedur na czas realizacji równoległych zadań. ZN Pol. Śl. S. Informatyka Nr 1356, Gliwice 1997.
9. Subieta K.: Ingres System zarządzania relacyjną bazą danych. Akademicka Oficyna Wydawnicza PLJ, Warszawa 1994.

10. Ingres/SQL Reference Manual for The Unix and VMS Operating Systems. Ingres Corporation, California 1991.

Recenzent: Prof. dr hab. inż. Tadeusz Czachórski

Wpłynęło do Redakcji 2 kwietnia 2002 r.

Abstract

The Remote Procedure Call (RPC) mechanisms are one of the most efficient tools based on message passing model. In order to increase their efficiency we presented a new method of how to use them with distributed shared memory. The proposed method (called RPCL) resembles to Linda computing model: The idea is to create shared memory spaces, called communication space (which correspond to Linda tuple spaces), on the remote machines. The communication space allows the server's and client's processes to exchange data in synchronic mode. Although the proposed method is not so sophisticated as the Linda tuple space mechanisms, the obtained results was satisfactory. To evaluate the new method, a parallel algorithm was presented and the results was compared to those obtained using other parallel tools.

The obtained results using RPCL, were the best comparing to other systems like Linda, PVM or traditional RPC.