ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1995
Seria: INFORMATYKA z. 28 Nr kol. 1270

Grzegorz KOWALCZYK

SYSTEM REALIZACJI ALGORYTMÓW RÓWNOLEGŁYCH W LOKALNYCH SIECIACH MASZYN UNIXOWYCH

Streszczenie. W artykule przedstawiona została opracowana przez autora biblioteka funkcji, służących do organizacji i nadzoru obliczeń wykonywanych współbieżnie przez komputery połączone siecią lokalną. Opracowane narzędzia wykorzystują protokół *Remote Procedure Call - RPC* oraz mechanizmy *Inter-Process Comunications - IPC*, dostępne praktycznie we wszystkich współczesnych odmianach systemu UNIX.

SYSTEM FOR PARALLEL ALGORITHMS REALISATION ON UNIX-BASED LOCAL AREA NETWORKS

Summary. The paper presents a tool (functions library) for parallel computations on UNIX-based local area networks. This method is based on *Remote Procedure Call - RPC* protocol and *Inter-Process Comunications IPC* mechanisms, which are implemented in many contemporary vresions of UNIX operating system.

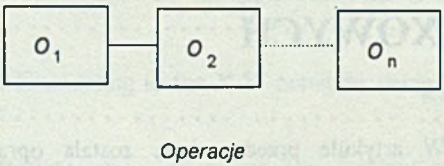
SYSTÈME DE RÉALISATION D'ALGORITHMES PARALLÈLES DANS LES RESEAUX LOCAUX DE MACHINES UNIX

Résumé. L'article présente une bibliothèque de fonctions développée par l'auteur. Les fonctions servent à organiser et surveiller les calculs réalisés en parallèle par dę calculateurs reliés par un reseau local. Les utils développés font appel au protocole *Remote Procedure Call - RPC* et aux mécanismes de *Inter-Process Communications - IPC*, disponibles sur pratiquement toutes les versions courantes du système Unix.

1. Wprowadzenie

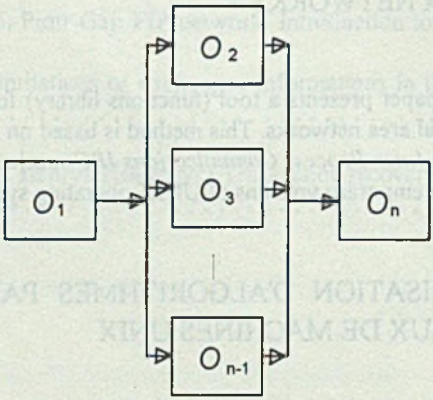
Jako algorytm rozumiany jest skończony, uporządkowany zbiór operacji, po wykonaniu których według ustalonego porządku na zbiorze danych otrzymamy rozwiązanie dowolnego zadania z określonej klasy zadań [1].

Gdy wykonywanie operacji następować będzie jedna po drugiej, tzn. wykonanie kolejnej operacji następować będzie po zakończeniu wykonywania poprzedniej, to algorytm taki nazywany będzie *algorytmem szeregowym*. Proces realizacji takich algorytmów można zobrazować schematem przedstawionym na rys. 1.



Rys. 1. Schemat realizacji algorytmów szeregowych
Fig. 1. Realization schema of serial algorithms

Jeżeli przewidywane jest współbieżne wykonywanie operacji opisanych algorytmem, to algorytm taki nazywamy algorytmem równoległym [1]. Przykład realizacji takiego algorytmu przedstawiony jest na rys. 2.



Rys. 2. Schemat realizacji algorytmu równoległego
Fig. 2. Realization schema of paralel algorithm

Realizacja algorytmów równoległych możliwa jest na komputerach o architekturze równoległej lub w sieci komputerów o dowolnej architekturze.

W niniejszym opracowaniu przedstawiony zostanie przykład systemu umożliwiającego realizację algorytmów równoległych w sieci komputerów pracujących pod kontrolą systemu operacyjnego SunOS 4.1.x (mutacja systemu UNIX). Mogą być wykorzystane dowolne połączenia sieciowe umożliwiające transmisję w protokole TCP/IP, jednak by uzyskać maksymalne przyśpieszenia realizacji zadań, należy wykorzystać możliwie szybkie sieci.

2. Model przetwarzania

W opracowanym systemie realizacji algorytmów równoległych w sieci komputerów przyjęto, że wykonywane równolegle fragmenty algorytmu opisane będą w postaci funkcji języka C, a ich uruchamianie, przekazanie argumentów i przejęcie zwracanego wyniku wykonania funkcji odbywać się będzie za pomocą mechanizmu systemowego noszącego nazwę Sun RPC *(Remote Procedure Call)* [2, 3, 5].

Zdalne uruchamianie procedur z wykorzystaniem RPC wymusza przyjęcie pewnych rozwiązań rozgałęziania procesów. W mechanizmie RPC maszyna wywołująca procedurę (funkcję) nosi miano klienta, a maszyna wykonująca tę procedurę - serwera. Ze względu na to, iż proces klienta nie może podjąć żadnej akcji, nim nie zakończy się realizacja zdalnie wywołanej funkcji, możliwe jest przyjęcie rozwiązania, w którym program serwera po nadejściu wywołania ze strony klienta będzie inicjował proces wykonujący fragment algorytmu, i nie czekając na zakończenie tego procesu, przekaże sterowanie do procesu klienta (nastąpi powrót z wywołania RPC). Zainicjowany na serwerze proces będzie przebiegał bez żadnej kontroli ze strony klienta i efekty jego działania nie będą przejęte jako wartość zwracana przez funkcję wywołaną poprzez RPC, lecz będą wymagały utworzenia innej drogi. Komputer - klient po odebraniu informacji o pomyślnym zainicjowaniu procesu na serwerze będzie mógł przejść do wykonywania kolejnego fragmentu algorytmu, który będzie przebiegał współbieżnie z procesem serwera, a w szczególności wywołać funkcję na kolejnym serwerze.

W zrealizowanym systemie przyjęto inny schemat rozgałęziania procesów. Na jednej z maszyn biorących udział w procesie przetwarzania uruchomiony zostanie program inicjujący wykonywanie algorytmu równoległego. Program ten, zwany dalej programem sterującym, przekazywał będzie wywołania funkcji realizujących fragmenty algorytmu do pamięci, z której inne współpracujące w procesie przetwarzania komputery, za pomocą programów pośredniczących, będą je mogły odebrać i wykonać współbieżnie. Program sterujący będzie mógł w tym czasie wykonywać inny fragment algorytmu lub oczekiwać na efekty wykonania funkcji realizowanych na pozostałych maszynach, a przekazywane przez dostępny dla wszystkich współpracujących maszyn obszar pamięci. Tworzone przez program sterujący wywołania funkcji, przeznaczone do wykonania na komputerach zdalnych, nosić będą miano

rozkazu tak zorganizowanej maszyny równoległej. Funkcje wywoływane zdalnie mogą posiadać tylko jeden parametr. Parametr ten musi być wskaźnikiem do dowolnego typu prostego lub złożonego (ograniczenia rozmiaru typu parametru omówione zostaną w dalszej części niniejszego opracowania). Pamięć, do której zwracane będą wyniki realizacji funkcji, nazywana będzie pamięcią rezultatów. Komputery przetwarzające pobierać będą rozkazy z pamięci rozkazów i zapisywać wyniki do pamięci rezultatów. Obszary pamięci rozkazów i rezultatów są rozłączne.

Z pamięci rezultatów wyniki wykonania funkcji odbierane będą przez program sterujący, który podda je dalszemu przetwarzaniu. Wyniki, zwracane przez funkcje realizowane zdalnie, przekazywane są przez wskaźnik do dowolnego typu prostego lub złożonego. Komputery przetwarzające, do realizacji zdalnie wywołanych funkcji, w pełni mogą wykorzystywać zasoby systemów lokalnych. Dostęp do wspólnych pamięci masowych (dysków) możliwy jest z wykorzystaniem sieciowego systemu plików NFS.

Z poziomu programu sterującego wstawianie rozkazów i pobieranie rezultatów odbywać się będzie w formie wywołania funkcji i możliwe będzie w dowolnym miejscu programu po zainicjowaniu systemu przetwarzania równoległego.

2.1. Organizacja pamięci rozkazów

Pamięć rozkazów zasadniczo ma organizację kolejki o regulaminie naturalnym. Program sterujący wstawia rozkaz na koniec kolejki, a program przetwarzający pobiera rozkaz znajdujący się na początku kolejki i wykonuje go. Możliwe jest jednakże oznaczenie każdego ze wstawianych rozkazów znacznikiem (numerem) i wskazanie poszczególnym procesorom przetwarzającym, o jakich znacznikach rozkazy mają wykonywać. W ten sposób powstanie pewna liczba kanałów w pamięci rozkazów, którymi przekazywane będą oznaczone rozkazy do wyznaczonych procesorów. W skrajnych przypadkach wszystkie rozkazy mogą zostać oznaczone tym samym znacznikiem (czyli numerem kanału), wówczas pamięć rozkazów faktycznie będzie miała organizację kolejki o regulaminie naturalnym, lub każdy z rozkazów może mieć unikatowy znacznik, wówczas kanałów będzie tyle ile rozkazów zostało wysłanych do pamięci. Procesory mogą wykonywać rozkazy odebrane z kanału o określonym numerze, z określonej grupy kanałów lub z dowolnego z kanałów. Pobierany z kanału jest rozkaz, który znalazł się w nim najwcześniej.

Kanały oznaczane są liczbami całkowitymi większymi od zera. Procesory przetwarzające podzielone są na klasy. Znacznikami klas są liczby całkowite. Każdy z procesorów może należeć tylko do jednej klasy. Procesory klasy *0* mogą wykonywać rozkazy przesyłane kanałem o dowolnym numerze, klasy *K* - rozkazy przesyłane kanałem *K*, a klasy *-K* rozkazy nadesłane kanałami *1...K*. Jeśli złożoność obliczeniowa pewnych, przeznaczonych do

równoległego wykonywania, procedur jest większa od pozostałych lub gdy chcemy, by wyniki wykonania pewnych procedur były uzyskiwane szybciej, a dysponujemy maszynami różniącymi się mocą obliczeniową lub w różnym stopniu obciążonymi innymi zadaniami, możemy tak przydzielić maszyny do klas i w taki sposób kierować wywołania procedur do odpowiednich kanałów, by procedury o większej złożoności obliczeniowej lub te, które powinny być wykonane w pierwszej kolejności, trafiały na komputery potencjalnie szybsze.

Zadanie pobierania rozkazów i przesyłania ich poprzez sieć realizują programy pośredniczące wykonywane na komputerze sterującym. Każdy z programów pośredniczących, zrealizowanych jako *RPC client*, przekazuje rozkazy do jednego komputera przetwarzającego. Programy pośredniczące przejmują także od maszyn przetwarzających rezultaty wykonania rozkazów i umieszczają je w pamięci rezultatów.

Pamięć rozkazów ma ograniczoną pojemność. Gdy proces sterujący przystępuje do wstawienia rozkazu, a pojemność pamięci rozkazów jest wyczerpana, jest on wstrzymywany do momentu zwolnienia pamięci poprzez pobranie rozkazu do wykonania.

Do pamięci rozkazów zapisywać może tylko program sterujący, z wyjątkiem sytuacji awaryjnych (np. wyłączenia komputera przetwarzającego w trakcie wykonywania rozkazu), gdy program pośredniczący może ponownie umieścić nie wykonany rozkaz w pamięci rozkazów.

2.2. Organizacja pamięci rezultatów

W pamięci tej przechowywane będą wyniki realizacji rozkazów. Programy pośredniczące odbierają wyniki zdalnego wykonania funkcji i zapisują je w pamięci rezultatów. Program sterujący może w dowolnym momencie pobrać wynik z pamięci rezultatów. Organizacja pamięci rezultatów jest odbiciem organizacji pamięci rozkazów. Jeśli rozkaz wysłany został kanałem K pamięci rozkazów, to rezultat będzie przesyłany kanałem K pamięci rezultatów. Program sterujący może wskazać, z którego kanału czy grupy kanałów chce pobrać wynik. Pojemność tej pamięci jest ograniczona. Gdy zapisany zostanie cały dostępny obszar, wówczas programy pośredniczące muszą wstrzymać zapisywanie wyników, aż do momentu gdy program sterujący odbierze jeden lub więcej rezultatów z pamięci.

Pamięci rozkazów i rezultatów zrealizowane są z wykorzystaniem mechanizmów systemowych *IPC messages* [4].

2.3. Format rozkazu

Rozkazy formowane są z poziomu programu sterującego i wstawiane do pamięci rozkazów. Jak już wcześniej wspomniano, wywoływane zdalnie funkcje mogą mieć tylko

jeden argument będący wskaźnikiem do typu prostego lub złożonego. Sformowany rozkaz mieć będzie format:

ID	Nazwa funkcji	Argument

ID - identyfikator rozkazu. Długość 4 bajty (liczba typu integer)
Nazwa funkcji - nazwa wywoływanej zdalnie funkcji (8 bajtów)
Argument - ciąg bajtów reprezentujących wartość argumentu wywoływanej zdalnie funkcji

Rys. 3. Format rozkazu
Fig. 3. Instruction format

Długość pola zarezerwowanego na argument jest określona wartością stałej MSIZE (znaczenie stałych omówione zostanie w dalszej części opracowania), i nawet w przypadku gdy rozmiar zmiennej przekazywanej jako argument funkcji będzie mniejszy od wartości MSIZE, każdy z rozkazów będzie zajmował w pamięci rozkazów stałą długość.

Każdy ze wstawianych przez program sterujący rozkazów musi mieć unikatowy ID. Jest to konieczne do identyfikacji wyników wykonania funkcji.

2.4. Format rezultatu

Przekazywany przez program pośredniczący do pamięci rezultatów wynik wykonania funkcji reprezentowany będzie w sposób następujący:

ID	Rezultat

ID - identyfikator rezultatu. Dług. 4 bajty (liczba integer)
Rezultat - ciąg bajtów reprezentujący wartość wyniku wykonania funkcji

Rys. 4. Format rezultatu
Fig. 4. Result format

Identyfikator rezultatu jest równy identyfikatorowi rozkazu, w wyniku wykonania którego rezultat został zwrócony. W polu "Rezultat" umieszczany jest ciąg bajtów reprezentujących wartość zwracanej zmiennej. Długość pola "Rezultat" jest stała i wynosi MSIZE.

2.5. Realizacja rozkazów

Pobrane przez program pośredniczący rozkazy są bazą do zorganizowania wywołania zdalnej funkcji za pomocą mechanizmów RPC. Po stronie procesorów przetwarzających, by obsłużyć wywołania RPC, muszą pracować programy przetwarzające pracujące jako serwery RPC. Programy te mogą realizować funkcje z wcześniej przygotowanej listy funkcji do wykonywania zdalnego.

3. Tworzenie aplikacji rozproszonych

Na podstawie przedstawionego modelu przygotowana została biblioteka funkcji języka C pozwalających programiście na tworzenie rozproszonych aplikacji i realizację w ten sposób algorytmów równoległych.

3.1. Funkcje inicjalizacji i zatrzymania systemu przetwarzania rozproszonego

3.1.1. Funkcja prll_init()

```
prll_cntrl  *prll_init()
```

Przed rozpoczęciem rozsyłania zadań na maszyny przetwarzające konieczne jest umieszczenie w programie wywołania bezparametrowej funkcji `prll_init()`. W efekcie jej działania zainicjowane zostaną pamięci rozkazów i rezultatów, uruchomione zostaną programy buforowania tych pamięci (`ibuffer` i `rbuffer`) oraz programy pośredniczące (`pclnt`). Zwrócony zostanie wskaźnik do struktury kontrolnej, która zdefiniowana jest następująco:

```
typedef struct  {
   int  ii; /* identyfikator wejścia pamięci rozkazów */
   int  io; /* identyfikator wyjścia pamięci rozkazów */
   int  ri; /* identyfikator wejścia pamięci rezultatów */
   int  ro; /* identyfikator wyjścia pamięci rezultatów */
   cl_pids *pids; /*lista zawierająca identyfikatory procesów pclnt
                  */
   int  ibpid; /* id procesu bufora rozkazów */
```

```
  int  rbpid; /* id procesu bufora rezultatów */
}prll_cntrl;

typedef struct pids{
  int  pid;
  struct pids  *ptr;
}cl_pids;
```

Po wykonaniu funkcji `prll_init()` mogą znaleźć się w programie wywołania funkcji wstawiającej rozkazy do pamięci oraz wywołania funkcji pobrania rezultatów, umożliwiające rozproszenie wykonywania funkcji realizujących algorytm równoległy na maszyny w sieci.

3.1.2. Funkcja prll_stop()

```
(void) *prll_stop(pc)
  prll_cntrl  *pc;
```

Po zakończeniu fazy równoległej programu należy przerwać działanie wszystkich programów obsługujących przetwarzanie rozproszone i zwolnić pamięć rozkazów i rezultatów. W tym celu należy użyć w programie wywołania funkcji `prll_stop()`.

3.2. Funkcje generowania rozkazu i pobrania wyniku: put() i get()

3.2.1. Funkcja put()

```
int  put(pc, id, chnll, fname, args)
  prll_cntrl *pc; /* wskaźnik do struktury kontrolnej zwrócony
                     przez funkcję prll_init */
  int  id; /* identyfikator rozkazu */
  int  chnll; /* numer kanału */
  char *fname; /* nazwa funkcji wywoływanej zdalnie */
  void *args; /* argument funkcji wywoływanej zdalnie */
```

Jako identyfikatory rozkazów mogą być użyte dowolne liczby typu `int`, numery kanałów mogą być liczbami dodatnimi, a nazwa wywoływanej funkcji może składać się co najwyżej z 8 znaków.

Jeśli rozkaz zostanie wstawiony do pamięci rozkazów, funkcja zwraca wartość 0, w wypadku niepomyślnej próby zapisu - zwracana jest wartość -1.

3.2.2. Funkcja get()

```
struct result  *get(pc, chnll)
   prll_cntrl *pc; /* wskaźnik do struktury kontrolnej */
   long chnll;  /*  numer  kanału,  z  którego  pobierany  będzie
rezultat*/

struct result {
   int  id;
   char res[MSIZE];
};
```

Funkcja zwraca wskaźnik do struktury `result`. W polu `res` zawarty jest ciąg bajtów reprezentujący wartość zwracaną przez funkcję, która została wykonana w wyniku realizacji rozkazu o identyfikatorze `id`. W sytuacji gdy pamięć rezultatów jest pusta, wywołanie funkcji `get` powoduje wstrzymanie wykonywania programu sterującego, aż do momentu gdy pobranie rezultatu z pamięci będzie możliwe. Gdy numer kanału będzie równy zero, pobrany zostanie rezultat przebywający najdłużej w pamięci w dowolnym kanale, gdy będzie równy *K*, pobrany zostanie rezultat przebywający najdłużej w kanale *K*, a gdy będzie równy -*K*, pobrany zostanie rezultat przebywający najdłużej w kanałach od *1* do *K*.

4. Instalowanie pakietu

Biblioteka funkcji oraz wszystkie, konieczne do prący systemu programy przygotowane są w postaci pliku o nazwie `pnm.tar`. Należy skopiować go do katalogu `/usr/local` i rozpakować poleceniem:

```
tar xvf pnm.tar
```

Zostanie utworzony katalog `/usr/local/pnm`, w którym znajdować się będą wszystkie niezbędne do tworzenia aplikacji rozproszonych narzędzia. Następnie z katalogu `/usr/local/pnm` należy wydać polecenie `pclntgen`.

Użytkownik, który będzie wykorzystywał zawarte w pakiecie funkcje i programy, musi dopisać katalog `/usr/local/pnm` do swojej ścieżki przeszukiwań.

5. Przygotowanie aplikacji

Tworzący aplikację rozproszoną musi przygotować blok programu sterującego oraz blok deklaracji funkcji, które będą uruchamiane zdalnie. W bloku programu sterującego muszą znaleźć się wywołania funkcji `prll_init()`, `put()`, `get()` i `prll_stop()`. W bloku deklaracji funkcji (odrębny plik) muszą znajdować się funkcje typu:

```
typ_1   *funkcja(arg)
   typ_2   *arg;
```

Typy `typ_1` i `typ_2`, są to dowolne typy proste lub typy złożone, których rozmiar nie przekracza wartości `MSIZE`, zdefiniowanej w pliku `/usr/local/pnm/pdef.h`. Deklaracja każdej funkcji poprzedzona musi być etykietą `BEGIN` i zakończona etykietą `END`.

5.1. Przykład

Poniżej przedstawiony zostanie program realizujący równoległy algorytm rozwiązywania układów równań liniowych metodą Cramera [1].

Program sterujący może mieć postać:

plik `r_liniowe.c`:

```
#include <stdio.h>
#include "/usr/local/pnm/host.c"  /* Niezbędne */
#include "deter.c"/* Plik deter.c zawiera funkcję obliczania
                     wartości wyznacznika macierzy*/
main(argc, argv)
   int  argc;
   char *argv[];

{
   int  n; /* zmienna przechowująca rozmiar układu równań */
   int  i, j, rtrn;
   double  wz;
   struct result   *r;
   struct argmnt {
      int  nn;  /* rozmiar układu rownań*/
```

```
    int  k;/*numer kolumny zamienianej wektorem wyrazów
            wolnych */
    double  dt;  /* wartość wyznacznika macierzy głównej */
    char f1[12]; /* nazwa pliku z elementami macierzy gł. */
    char f2[12]; /* nazwa pliku z elementami wektora wyrazów
                    wolnych */
  } *a;
  double  *dd;
  FILE *f;
  prll_cntrl *pc; /* wskaźnik do struktury kontrolnej */

  a = (struct argmnt *) malloc(sizeof(struct argmnt));
  n = atoi(argv[1]);
  wz = d(argv[2], argv[3], n, 0);/* obliczenie wyzn. głównego */
  /* funkcja d() zadeklarowana jest w pliku deter.c. Jej
     argumentami są:
     nazwa pliku z elementami macierzy głównej,
     nazwa pliku z elementami wektora wyrazów wolnych,
     rozmiar układu równań,
     numer kolumny macierzy głównej zamienianej wektorem wyrazów
     wolnych (0 oznacza brak wymiany) */
  if (wz == 0) {
     printf("Wyznacznik glowny = 0\n");
     exit(1);
  }
  f = fopen("wyniki","a");
  a->dt = wz;
  pc = prll_init(); /* inicjacja przetwarzania rozproszonego */
  strcpy(a->f1, argv[2]);
  strcpy(a->f2, argv[3]);
  a->nn = n;
  for (j = 1; j <= n; j++) {
     a->k = j;
     do
        rtrn = put(pc, j, 1,"prwstk", a);
/*wstawienie do pamięci rozkazów wywołania funkcji prwstk */
     while (rtrn == -1);
  }
```

```
  for (i = 1; i <= n; i++) {
    do
      r = (struct result *) malloc(sizeof(struct result));
    while (w == NULL);
    r = get(pc, 0); /* pobranie rezultatu */
    dd = (double *)r->res;
    fprintf(f,"x(%d) = %f\n",r->id,*dd);
    free(r);
    free(r->res);
  }
  prll_stop(pc); /* zakończenie pracy równoległej */
  free(a);
  fclose(f);
  exit(0);
}
```

Zawartość pliku z deklaracjami funkcji wywoływanych zdalnie będzie następująca:

plik deklaracje_funkcji:

```
#include <stdio.h>
#include "deter.c"

BEGIN
double *prwstk(a)
  struct argmnt *a;
{
  double *x;

  x = (double *)malloc(sizeof(double));
  *x = d(a->f1,a->f2,a->nn,a->k)/a->dt;
  return(x);
}
END
```

Po przygotowaniu pliku z kodem programu sterującego i pliku z deklaracjami funkcji należy w katalogu, w którym znajdują się ww. pliki,wydać polecenia:

```
fgen deklaracje_funkcji
```

```
pservgen
```

oraz

```
cc r_liniowe.c -o r_liniowe
```

Powstały w wyniku działania polecenia `pservgen` program `pserv` należy uruchomić na wszystkich maszynach przetwarzających. Program `pserv` jest programem serwera RPC i jego uruchomienie (tzn. napisanie jego nazwy w wierszu poleceń i naciśnięcie klawisza Enter) spowoduje jedynie jego zarejestrowanie. Program ten nie będzie obciążał systemu, aż do momentu nadejścia wywołania ze strony programu klienta RPC.

Ostatnią czynnością konieczną przed uruchomieniem aplikacji jest utworzenie pliku, w którym umieszczone zostaną nazwy maszyn przetwarzających *(hostnames)* wraz z numerami klas, do jakich zostały przypisane. Plik musi nosić nazwę `.servers` i znajdować się w katalogu, z którego uruchamiana będzie aplikacja. Przykładowa zawartość pliku `.servers`:

```
serwer1 0
serwer2 0
serwer3 0
```

Na wszystkich maszynach, których nazwy zostały umieszczone w tym pliku, muszą być uruchomione programy `pserv`. W wypadku gdy na którejś z maszyn wymienionych w tym pliku program `pserv` nie będzie funkcjonował, system poprawnie podejmie realizację programu rozproszonego na pozostałych maszynach. Mechanizmy RPC pozwalają, by maszyna była jednocześnie klientem i serwerem RPC, więc możliwe jest, by komputer, na którym realizowany jest program sterujący, także wykonywał funkcje przeznaczone do wykonywania zdalnego. Należy jednak wziąć pod uwagę to, że programy buforowania pamięci rozkazów i rezultatów oraz programy pośredniczące w znacznym stopniu obciążają system.

Gdy wykonaliśmy wszystkie wyżej opisane czynności oraz przygotowaliśmy pliki zawierające wartości elementów macierzy głównej układu i macierzy wyrazów wolnych, możemy przystąpić do uruchomienia przykładowej aplikacji poleceniem:

```
r_liniowe 100 m.glowna m.w_wolnych
```

(parametry: rozmiar układu, plik z elementami macierzy głównej, plik z elementami wektora wyrazów wolnych)

W efekcie wykonania programu w pliku `wyniki` powinny znajdować się wartości pierwiastków równań.

6. Obsługa sytuacji awaryjnych

System potrafi zareagować na większość sytuacji awaryjnych, jakie mogą wystąpić w trakcie realizacji aplikacji rozproszonej.

W sytuacji gdy rozkaz pobrany z pamięci rozkazów przez program pośredniczący nie może zostać przesłany do komputera przetwarzającego, jest zwracany do pamięci, a użytkownik zostaje poinformowany komunikatem. Jeśli w systemie pracuje inny komputer przetwarzający mogący pobrać zawrócony rozkaz (numery kanałów!), zadanie powinno zostać ukończone pomyślnie.

Czas między wysłaniem przez program pośredniczący rozkazu do komputera przetwarzającego a odebraniem przez program pośredniczący wyników wykonania jest ograniczony przez wartość *RPC time out* (powrót z wywołania procedury musi nastąpić w tym czasie, jeśli nie nastąpi, to mechanizmy *Sun RPC* określą tę sytuację jako błędną). Wartość *RPC time out* jest równa stałej T_OUT zdefiniowanej w pliku `/usr/local/pnm/pdef.h`. Wartość T_OUT powinna być tak oszacowana, by czas wykonywania najbardziej złożonej funkcji na najwolniejszym z komputerów był od niej mniejszy. Jeśli taki czas okaże się zbyt krótki na wykonanie rozkazu na danym procesorze, rozkaz zostanie zapisany ponownie do pamięci rozkazów (użytkownik zostanie powiadomiony komunikatem), co stwarza możliwość, że zostanie pobrany do wykonania na innym, szybszym procesorze. Nie istnieje jednak zabezpieczenie przed ponownym skierowaniem rozkazu na procesor, z którego został zawrócony.

By zwiększyć pewność działania systemu, istotne jest poprawne oszacowanie *time out*'u oraz takie przydzielenie procesorów przetwarzających do klas i generowanych zadań do kanałów, by każdy rozkaz mógł być wykonywany na alternatywnym komputerze przetwarzającym.

7. Ustalenie wartości stałych

W pliku `/usr/local/pnm/pdef.h` znajdują się m. in. deklaracje stałych. Zmiana wartości niektórych z nich pozwala zmienić właściwości systemu. Rola niektórych stałych została przedstawiona w tekście powyżej.

Zawartość pliku `pdef.h` jest następująca (fragment):

```
#define NLEN 8     /* Długość pola nazwy funkcji.          */
```

```
#define IDLEN 4    /* Długość pola identyfikatora = sizeof(int) */
#define MSIZE 64   /* Długość pola argumentu|rezultatu.      */
#define BUFLEN 2000  /* Pojemność bufora                 */
#define T_OUT 100 /* Wartość RPC timeout  [s]     */
#define MAXMSGLEN 76 /* MAXMSGLEN = NLEN + IDLEN + MSIZE
                        Zmienić wraz ze zmianą składowych    */
```

Znaczenie niektórych stałych:

NLEN - wskazuje, z ilu znaków może składać się nazwa wywoływanej zdalnie funkcji. Nadanie większej wartości nie jest wskazane.

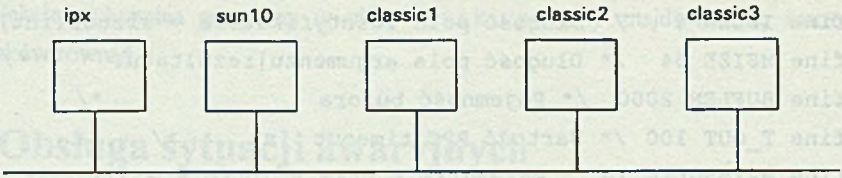
MSIZE - maksymalny rozmiar typu przesyłanego jako argument funkcji i typu zwracanego przez funkcję. Musi być tak dobrana, by wartość stałej MAXMSGLEN nie przekroczyła wartości stałej MSGMNB zdefiniowanej w /usr/include/sys/msg.h i wynoszącej standardowo dla systemu SunOS 4.1.1 2048 bajtów [4].

BUFLEN - pojemność pamięci rozkazów i rezultatów wyrażona ilością rozkazów. Można wysłać BUFLEN rozkazów bez odbierania rezultatów wykonania funkcji.

Znaczenie pozostałych stałych zostało objaśnione w tekście.
Zmiana wartości jakiejkolwiek stałej wymaga ponownej kompilacji programów systemu poleceniem pclntgen oraz pservgen.

8. Efekty wykorzystania systemu

Poniżej przedstawione zostaną wyniki eksperymentów przeprowadzonych z wykorzystaniem omówionego wyżej systemu. Eksperymenty prowadzone były w lokalnej sieci komputerów Sun, komunikujących się w standardzie Ethernet. Pierwszy z eksperymentów wykorzystywał przedstawiony w pracy program równoległego rozwiązywania układów równań liniowych metodą Cramera. By wyznaczyć uzyskiwane dzięki zrównolegleniu przyspieszenia, przeprowadzono także pomiary szeregowej realizacji algorytmu na tych samych komputerach. Wykorzystano pięć komputerów Sun noszących nazwy: ipx, sun10, classic1, classic2 i classic3.



Rys. 5. Wykorzystana w eksperymentach sieć lokalna
Fig. 5. UNIX-based LAN for experiments

Komputery classic1 - 3 oraz ipx posiadają taką samą moc obliczeniową, komputer sun10 jest ok. 2.2 raza szybszy.

Poniższa tabela zawiera czasy rozwiązania układów równań liniowych o różnych rozmiarach równolegle w sieci komputerów oraz szeregowo na najszybszym z nich.

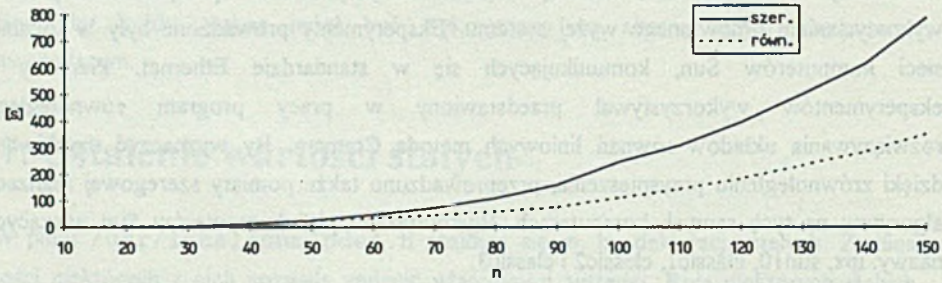
Tabela 1

Wyniki eksperymentu

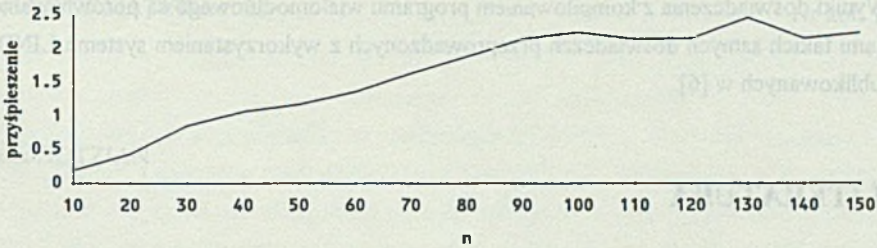
n	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
szer.	0.6	2	6	14	26	46	74	110	159	246	308	396	501	614	787
równ.	3	4.5	7	13	22	33.5	45	58	74	109	143	183	233	283	351
przysp.	0.2	0.44	0.86	1.07	1.18	1.37	1.64	1.89	2.15	2.25	2.15	2.16	2.46	2.16	2.24

n- rozmiar układu równań

Czasy obliczeń podane są w sekundach, a przyspieszenie jest stosunkiem czasu obliczeń szeregowych do czasu obliczeń równoległych. Obliczenia prowadzone były przy losowym obciążeniu sieci i komputerów innymi zadaniami. Przedstawione w tabeli wyniki są średnią z dwóch eksperymentów.



Rys. 6. Czasy obliczeń szeregowych i równoległych w funkcji rozmiaru układu równań
Fig.6. Time values of parallel and serial computations depending on dimension of linear equations system



Rys. 7. Uzyskane przyśpieszenia w funkcji rozmiaru układu równań
Fig. 7.Acceleration depending on linear equations system dimension

Daje się zauważyć wzrost przyspieszenia wraz ze wzrostem czasu wykonywania wywołanej zdalnie pojedynczej funkcji. Zwiększenie czasu wykonywania funkcji w tym wypadku związane jest ze wzrostem rozmiaru układu równań. Maksymalne, teoretyczne, przyśpieszenie, wynikające ze zsumowania mocy procesorów, możliwe do uzyskania w wykorzystanej sieci komputerów wynosi ok. 2.8. W przedstawionym przykładzie, dla większych rozmiarów układu równań, uzyskano wartość przyspieszenia równą ok. 80% wartości maksymalnej.

Operacje wykonywane równolegle, na jakie rozbijamy algorytm, nie mogą być operacjami, których czas realizacji jest niewielki w stosunku do czasu potrzebnego na zorganizowanie przesłania wywołania funkcji na zdalną maszynę i odebrania wyników jej wykonania. Nie powinny być to operacje elementarne typu sumy dwóch liczb czy ich ilorazu. Rozbicie algorytmu na zbyt proste operacje i wykonywanie ich równolegle w tym systemie przyniesie efekt odwrotny od oczekiwanego. Związane jest to z opóźnieniami wnoszonymi przez programy umożliwiające pracę równoległą oraz opóźnieniami wnoszonymi przez sieć.

Drugi z przeprowadzonych eksperymentów polegał na równoległej kompilacji programu w języku C. Kompilowany program składał się z 14 modułów kompilowanych oddzielnie, a następnie linkowanych w jeden program wykonywalny. W zrealizowanym programie wykorzystano możliwość równoległej kompilacji. Programy źródłowe oraz efekty ich kompilacji - pliki typu *object* przekazywane były poprzez sieciowy system plików NFS. Komputer sterujący wysyłał do komputerów przetwarzających polecenia skompilowania poszczególnych modułów i czekał na sygnały zakończenia operacji. Po skompilowaniu wszystkich modułów program sterujący przechodził w fazę szeregową i odbywało się linkowanie plików typu *object*. Kompilowany był program `gzip`.

Czas kompilacji i linkowania na jednym komputerze: 38.6 s.
Czas kompilacji i linkowania w sieci: 22 s.

Uzyskane przyspieszenie: 1.75

Wyniki doświadczenia z kompilowaniem programu wielomodułowego są porównywalne z efektami takich samych doświadczeń przeprowadzonych z wykorzystaniem systemu LINDA, a opublikowanych w [6].

LITERATURA

[1] Węgrzyn S., Czachórski T., Vidal P., Gille J.Ch.: Algorytmy i procesy równoległe w informatyce, w biologicznych systemach rozwojowych i w systemach masowej obsługi, GRANT KBN 3 P 406 011 04.

[2] Comer D., Stevens D.L.: Internetworking with TCP/IP vol. III, Prentice-Hall 1993

[3] Weiss Z., Gruźlewski T.: Programowanie współbieżne i rozproszone, WNT, W-wa 1993

[4] SunOS 4.1.1 Programming Utilities & Libraries, Sun Microsystems 1990.

[5] SunOS 4.1.1 Network Programming Guide, Sun Microsystems 1990.

[6] Bajerski P:. Realizacja algorytmów w systemie sterowanym przepływem argumentów w sieci lokalnej, Zeszyty Naukowe Politechniki Śląskiej, Seria Informatyka z. 26 1994.

Recenzent: Dr inż. Krzysztof Nałęcki

Wpłynęło do Redakcji 28 lipca 1994 r.

Abstract

The article presents method of performing parallel computations on Unix-based networks. The method is based on RPC protocol and IPC mechanizm called messages. RPC is a high-level communications paradigm which allows to design programs within client-server network model. With RPC, the client makes a procedure call which sends request to the server. When the server performs request the client suspends its execution. Together with added mechanizms RPC can distribute tasks and colect results of the computations.

Such a method was implemented and tested. The presented functions can be performed on the network according to the program prepared by a user.