ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1994

Seria: INFORMATYKA z. 27 Nr kol. 1259

Mohamed DAGHESTANI

IMPLEMENTACJA PAKIETU KOMUNIKACYJNEGO DLA TRANSPUTERÓW

Streszczenie. W pracy przedstawiono implementację pakietu komunikacyjnego dla systemu wieloprocesorowego złożonego z transputerów połączonych w dowolny sposób. Jego funkcjami są wysyłanie wiadomości do wybranego procesu lub rozsyłanie do wszystkich procesów. Proces komunikacyjny wybiera zawsze najkrótszą drogę przesyłania wiadomości i jest zabezpieczony przed wzajemną blokadą. W ramach pracy przeprowadzono badania eksperymentalne, w czasie których porównano zaimplementowany pakiet z systemem Express pod względem czasów przesyłania danych.

AN IMPLEMENTATION OF A ROUTER FOR TRANSPUTERS

Summary. In this paper an implementation of a router for any topology of transputers is presented. This router allows to send messages to any process or to broadcast it. The router chooses always the shortest path to the destination using a deadlock free routing. The routing performance of the message sending was tested and compared with the system Express.

IMPLÉMENTATION D'UN ROUTEUR POUR TRANSPUTERS

Résumé. Dans cet article on a présenté une implémentation d'un routeur pour les systèmes multiprocesseurs composés de transputers connectés d'une façon quelconque. Ce système permet d'envoyer un message à un processus déterminé ou de le distribuer à tous les processus. Le routeur choisi toujours le parcours le plus court et utilise un routage non blocant. On a testé le temps de transmission de message du routeur. Ces résultats ont été comparés avec ceux obtenus avec le système Express.

1. Wprowadzenie

Program równoległy w modelu CSP [6] składa się z procesów komunikujących się ze sobą za pomocą kanałów. Transputer [2,5] jest sprzętową realizacją tego modelu. Procesy wykonywane na różnych transputerach nie mogą korzystać ze wspólnej pamięci. Komunikacja jest synchroniczna, tzn. że proces wysyłający wiadomość zawiesza swoje wykonywanie do chwili odebrania wiadomości przez adresata. Procesy mogą być wykonywane na różnych transputerach, a każdy transputer może wykonywać kilka procesów. Liczba tych procesów jest ograniczona jedynie dostępną pamięcią operacyjną w transputerze. Rozróżnia się dwa rodzaje kanałów dla przesyłania danych: kanały wewnętrzne i kanały zewnętrzne. Kanały wewnętrzne służą do wymiany wiadomości między procesami wykonywanymi na tym samym transputerze. Za pomocą kanałów zewnętrznych można przesyłać dane między procesami wykonywanymi na różnych transputerach. Transputer wyposażony jest w cztery łącza służące do komunikowania się z innymi transputerami. Łącza składają się z dwóch kanałów, po jednym w każdym kierunku. Transmisja danych na tych łączach odbywa się za pomocą kanałów DMA. Transputer może więc mieć co najwyżej czterech sąsiadów, natomiast sposób łączenia transputerów jest dowolny. Z punktu widzenia programisty kanały wewnętrzne i zewnętrzne niczym się nie różnią, nawet na poziomie języka maszynowego. Sposób rozmieszczenia procesów w transputerze jest określony dopiero na etapie konfigurowania programu równoległego. Ograniczenie na liczbę kanałów zewnętrznych sprawia, że komunikujące się procesy nie mogą zawsze być połączone bezpośrednio. Należy więc często dołączyć do programu użytkowego fragment programu służący do przekazywania wiadomości między transputerami. Programy są ponadto często tak napisane, że zmiana konfiguracji sprzętowej, na którym program równoległy jest wykonywany, prowadzi do znacznej modyfikacji programu.

Funkcją pakietu komunikacyjnego (ang. router) jest wysyłanie wiadomości do dowolnego innego procesu przez podanie jedynie jego identyfikatora. Identyfikatory nadawcy i odbiorcy definiują kanał wirtualny, przez który wiadomość jest przesyłana. Zmiana konfiguracji sprzętowej systemu transputerowego nie wymaga żadnej zmiany w programie użytkownika. Proces komunikacyjny wykonywany na każdym transputerze wybiera odpowiednią, najkrótszą drogę do docelowego procesu, korzystając z tablic tworzonych w czasie konfigurowania programu. Proces komunikacyjny realizuje także rozsyłanie (ang. broadcast) wiadomości do wszystkich pozostałych procesów i zabezpieczony jest przed wzajemną blokadą procesów (ang. deadlock) [1,4].

Zaimplementowany pakiet komunikacyjny jest w dużej mierze oparty na systemie TINY opisanym w pracy [3] i został napisany w języku Parallel C [9] firmy 3L. W literaturze można się spotkać z innymi koncepcjami pakietów komunikacyjnych dla transputerów. W pracy [8] porównano niektóre z nich. W Polsce został wykonany pakiet [7] współpracujący z kompilatorem języka Parallel C firmy Inmos (wersje Parallel C firmy Inmos i 3L nie są zgodne). W niniejszej pracy porównano zaimplementowany pakiet z komercyjnym systemem Express [10] pod względem czasów przesyłania danych.

2. Ogólny opis pakietu komunikacyjnego

2.1. Wstęp

W każdym transputerze wykonywany jest proces komunikacyjny *router* służący do wymiany wiadomości pomiędzy procesami wykonywanymi na tym samym transputerze oraz do pośredniczenia w przesyłaniu wiadomości pomiędzy różnymi transputerami. Proces *router* przejmuje kontrolę nad wszystkimi łączami transputera. Wszystkie procesy (z pewnymi wyjątkami) muszą być "połączone" z procesem *router* i mają unikatowe numery identyfikacyjne w całym systemie transputerowym. Procesy nie połączone z procesem *router* (tzw. lokalne) mogą komunikować się jedynie z procesami wykonywanymi na tym samym transputerze poprzez własne kanały wewnętrzne. Na jednym transputerze może być wykonywanych kilka procesów korzystających z procesu komunikacyjnego. Podstawowym problemem, jaki należy rozwiązać w pakiecie komunikacyjnym, jest utworzenie tablic służących do wyboru łącza fizycznego dla dowolnego procesu docelowego. Zakładamy, że struktura połączeń procesów jest statyczna, tzn. że w czasie wykonywania programu nie mogą powstać nowe procesy. Tablice są utworzone na etapie konfigurowania programu za pomocą programu *prouter.exe*. Wiadomości są dzielone na pakiety. Pozwala to przesyłać wiadomości o dowolnej długości niezależnie od rozmiaru buforów. Wykonano dwie wersje pakietu różniące się metodą rezerwowania buforów. W pierwszej wersji rezerwuje się bufory w miarę "przemieszczania się" wiadomości w sieci transputerów. Natomiast w drugiej wersji rezerwuje się najpierw bufory w każdym transputerze leżącym na drodze przesyłania danych. Dopiero później następuje wysłanie pierwszego pakietu wiadomości. Wykonano ponadto wersję pakietu komunikacyjnego nie chroniącą przed wzajemną blokadą procesów. W punkcie 3.2 różnice między tymi wersjami zostaną szczegółowo omówione.

2.2. Zapobieganie wzajemnej blokadzie

Niech będą dane procesy $A_1,\ldots,A_k$. Wzajemna blokada procesów (ang. deadlock) występuje wtedy, gdy proces A_n czeka na zwolnienie bufora procesu A_{n+1} dla $n=1,\ldots,k$-1, a proces A_k czeka na zwolnienie bufora procesu A_1 [1]. Utwórzmy graf G, w którym wierzchołki przedstawiają transputery, a krawędzie - łącza między nimi. Wzajemna blokada może wystąpić wtedy, gdy graf G zawiera cykl. Załóżmy, że największa odległość między dwoma dowolnymi wierzchołkami grafu, tzw. *średnica grafu*, wynosi M. Podzielmy zbiór buforów w transputerze na M klas, B_1, B_2, ..., B_M. Jeśli przy wysyłaniu wiadomości przechowanej w buforze klasy B_l w transputerze A_n do bufora klasy B_m transputera A_{n+1}, zawsze będzie spełniona zależność $m>l$, to wzajemna blokada procesów nigdy nie wystąpi [1]. Ta metoda unikania wzajemnej blokady, choć prosta, jest dosyć kosztowna pamięciowo, gdyż w najgorszym przypadku należy podzielić zbiór buforów na p-1 klas, gdzie p oznacza liczbę transputerów. Klasy buforów mogą być nierównomiernie obciążone.

3. Opis procesu komunikacyjnego

Proces komunikacyjny jest wykonywany w każdym transputerze systemu. Proces ten jest połączony za pomocą kanałów z procesami wykonywanymi na tym samym transputerze oraz z procesami komunikacyjnymi sąsiednich transputerów. Zadaniem procesu komunikacyjnego jest przygotowanie tablic służących do wyboru dalszej drogi dla wiadomości oraz utworzenie wątku (ang. thread) [9] dla każdego kanału. Przygotowanie tablic odbywa się na podstawie danych zawartych w pliku *opis* utworzonym przez program *prouter*. Wątek jest procesem tworzonym dynamicznie przez proces nadrzędny. Wątki mogą współdzielić dane procesu nadrzędnego. Wątki wykonywane są, podobnie jak procesy, w jednym z dwóch trybów (priorytetów) pracy transputera: *pilny* (ang. urgent) lub *normalny* (ang. not urgent). Wykonywanie wątku (lub procesu) w trybie pilnym może być przerwane jedynie wtedy, gdy proces oczekuje na realizację komunikacji. Wątki w trybie normalnym wykonywane są metodą podziału czasu. Wątki utworzone przez proces komunikacyjny są wykonywane w trybie pilnym. Jest to niezbędne dla poprawnego działania procesu komunikacyjnego oraz umożliwia dostęp do współdzielonych danych bez użycia semaforów. Wyróżniamy 4 rodzaje wątków w zależności od tego, czy wątek obsługuje kanał wejściowy lub wyjściowy oraz czy kanał ten jest wewnętrzny, lub zewnętrzny. Każdemu wątkowi jest przyporządkowany numer służący do wyznaczenia obszaru jego

danych przez inne wątki. Z każdym wątkiem wyjściowym związane są listy buforów czekających na realizację komunikacji. Dla każdej klasy buforów tworzona jest niezależna lista. Wątek wyjściowy wykonuje pętlę, w której na początku wybiera pierwszy bufor swojej listy dla danej klasy buforów. W przypadku gdy nie ma takiego bufora, wątek zawiesza swoje wykonywanie do momentu, gdy jakiś wątek wejściowy dołączy bufor do jego listy. Po wysłaniu wiadomości przez kanał bufor zostanie zwrócony do listy wolnych buforów, pod warunkiem że nie należy do listy innego wątku wyjściowego (w przypadku rozsyłania). Wątki wejściowe realizują odczytanie danych z kanału wejściowego do bufora oraz dołączanie tego bufora do kolejki jednego lub kilku wątków wyjściowych.

3.1. Struktury danych

Zbiór buforów jest podzielony na *M* klas. Klasa o numerze 0 jest zarezerwowana dla wątków wejściowych wewnętrznych i musi zawierać przynajmiej tyle buforów, ile jest procesów wykonywanych w transputerze. W przeciwnym razie, co łatwo zauważyć, mogłoby dojść do blokady. Struktura bufora jest przedstawiona poniżej :

```
struct bufor {
    struct typNagłówka {
        typNagłówka typ ;
        int klasa, skad, dokad, długość ;
    } nagłówek ;
    int wIluListach ;
    struct bufor *łącznik[maks. liczba wątków wyjściowych] ;
    struct bufor *następnyWolny ;
    char *informacjeDoWysłania ; } ;
```

Wysłanie wiadomości jest poprzedzone wymianą nagłówków służących do żądania bufora danej klasy oraz do zezwolenia na wysłanie pakietu wiadomości. Istnieją 3 typy nagłówków (określane przez pole *typ)*:

- Żądania bufora. Zostaje wysłany przez *wątek wyjściowy zewnętrzny* przed wysłaniem pakietu. Po wysłaniu żądania wątek czeka na potwierdzenie, aby móc wysłać pakiet.
- Zezwolenia na wysłanie wiadomości danej klasy. Nagłówki tego typu mają odrębną klasę. Nie mogą więc spowodować blokady procesu komunikacyjnego.
- Wiadomości. Bezpośrednio po tym nagłówku zostanie wysłana wiadomość.

Pole *wIluListach* określa liczbę list wątków wyjściowych, do których bufor należy. Bufory mogą być współdzielone w czasie rozsyłania. Pole *łącznik* służy do połączenia buforów tego samego wątka wyjściowego w listę. Każdy wątek wyjściowy korzysta z innego elementu tablicy *łącznik*. Pole *następnyWolny* służy do połączenia wolnych buforów tej samej klasy w listę. Pole *informacjeDoWysłania* jest wskaźnikiem do bufora o długości określonej przez program *prouter* i zawiera dane do wysłania. Bufory klasy 0 zawierają wskaźnik do obszaru procesu użytkownika. Bufor użytkownika staje się buforem procesu komunikacyjnego. Pozwala to unikać zbędnego kopiowania danych.

Z każdą klasą związana jest struktura o postaci:

```
struct daneKlasyBuforów {
    struct bufor *pierwszyWolny ;
    typKolejki1 kolejkaŻądań ; } ;
```

Pierwsze pole wskazuje na początek listy wolnych buforów danej klasy. Pole o strukturze kolejki *kolejkaŻądań* służy do zapamiętania żądań buforów z innych transputerów, dla których nie było wolnego bufora. Procedura zwolnienia bufora sprawdza najpierw tę kolejkę i wysyła zezwolenie do odpowiedniego transputera, jeśli się okazało, że nie jest pusta. Struktura kolejki jest parokrotnie używana w implementacji pakietu komunikacyjnego. Struktura ta wraz z mechanizmem wewnętrznego szeregowania procesów w transputerze nie dopuszcza do zagłodzenia któregokolwiek z procesów systemu.

Z każdym wątkiem związana jest struktura o postaci:

```
struct daneWątku {
    typKolejki2 KolejkaKlas ;
    struct {
        struct bufor *pierwszyBufor ;
        struct bufor *ostatniBufor ;
        struct bufor *zarezerwowanyBufor ;
    } daneKlasy[maks. liczba klas] ;
    CHAN synchr ; } ;
```

Pole *KolejkaKlas* jest wykorzystywane jedynie przez wątki wyjściowe. Zawiera numery klas, których kolejki należy badać. Początki i końce listy dla danej klasy buforów są wskazywane przez pola *pierwszyBufor* i *ostatniBufor*. Bufory są zawsze dołączane na końcu listy przez wątki wejściowe. W ten sposób zapewniona jest prawidłowa kolejność odbioru danych. Po wysłaniu żądania bufora do następnego transputera numer danej klasy

jest usuwany z *KolejkiKlas* i zostanie ponownie do niej wprowadzony dopiero po otrzymaniu zezwolenia z następnego transputera przez odpowiedni wątek wejściowy. Pole *zarezerwowanyBufor* jest używane jedynie przez wątki wejściowe zewnętrzne po rezerwacji bufora danej klasy i przed otrzymaniem danych. Kanał wewnętrzny *synchr* służy do zawieszenia wykonywania wątku wyjściowego w przypadku braku pakietu lub nagłówka do wysłania. Zostanie on obudzony przez wątek wejściowy po dołączeniu bufora do jego listy.

3.2. Metoda przesyłania danych

Dostęp do fizycznych łączy transputerów jest multipleksowany pomiędzy wiadomościami przechowanymi w buforach różnych klas. Ma to oczywiście ujemny wpływ na czas wysłania wiadomości. Cykl wysłania wiadomości składa się z wysłania nagłówka żądania bufora danej klasy, oczekiwania na zezwolenia wysłania wiadomości, wysłania nagłówka wiadomości, po którym następuje wysłanie właściwej wiadomości. Proces komunikacyjny dzieli wiadomości nie mieszczące się w buforze na pakiety. Pozwala to nie tylko oszczędzać pamięć, lecz przyspiesza to przesyłanie większych wiadomości, gdyż występuje równoległe przesyłanie pakietów na poszczególnych etapach drogi.

Zaimplementowano trzy wersje pakietu komunikacyjnego. W pierwszej z nich po zarezerwowaniu bufora w kolejnym transputerze zostaje wysłany pierwszy pakiet wiadomości. Wysyłanie kolejnej porcji wiadomości może się odbywać dopiero po otrzymaniu zezwolenia z kolejnego transputera. Natomiast w drugiej wersji na całej drodze przesyłania danych są najpierw zarezerwowane bufory. Dopiero wątek wyjściowy wewnętrzny połączony z procesem docelowym wysyła zezwolenie na przesyłanie danych. Potwierdzenie to zostanie przekazane wszystkim transputerom. Po otrzymaniu potwierdzenia wysyłany jest pierwszy pakiet wiadomości. Natomiast następne pakiety zostają przez wątek wejściowy zewnętrzny od razu kopiowane z wejściowego łącza fizycznego do wyjściowego kanału (wewnętrznego lub zewnętrznego) nie korzystając z wątku wyjściowego, ani nie czekając na zezwolenie z następnego transputera. Łącza fizyczne są wtedy rezerwowane dla pakietów jednej wiadomości. Jak wykazują wyniki badań, pozwala to osiągać krótszy czas przesyłania danych. Druga metoda ma ponadto tę zaletę, że wiadomości między procesami użytkownika mogą być przesłane po różnych (najkrótszych) drogach. W pierwszej wersji pakietu byłoby to nieefektywne, gdyż wymagałoby to wprowadzenia potwierdzenia przyjęcia danych przez proces docelowy lub wprowadzenia liczników pakietów wiadomości tak, aby zapewnić prawidłową kolejność odbioru danych. W drugiej metodzie wymagane potwierdzenie z procesu docelowego jest automatyczne. Druga metoda może się okazać nieefektywna w przypadku rozsyłania wiadomości, gdyż wymaga

potwierdzenia z każdego procesu połączonego z procesem komunikacyjnym przed rozpoczęciem transmisji danych. Trzecia wersja pakietu komunikacyjnego stosuje podobną metodę jak druga wersja, z tą różnicą że istnieje tylko jedna klasa buforów. Metoda ta nie chroni przed wzajemną blokadą procesów. Łącza fizyczne nie są już multipleksowane między buforami różnych klas. Upraszcza to znacznie protokół przesyłania danych. Niepotrzebne stają się nagłówki potwierdzenia.

4. Sposób korzystania z pakietu komunikacyjnego

Przygotowując program równoległy w języku Parallel C [9], należy każdy proces kompilować osobno oraz utworzyć plik konfiguracyjny zawierający:

- opis fizycznych połączeń transputerów,
- definicje procesów, określajace wymagania pamięciowe oraz liczbę kanałów wejściowych i wyjściowych procesu,
- sposób rozmieszczenia procesów w transputerach,
- definicje kanałów między procesami.

Na tej podstawie specjalny systemowy program *config* tworzy wersję ładowalną programu równoległego.

4.1. Przetwarzanie pliku konfiguracyjnego

Został napisany program *prouter.exe* służący do przygotowania pliku konfiguracyjnego przy korzystaniu z pakietu komunikacyjnego. Wywołuje się go następująco:

prouter.exe <*wstępny plik konfiguracyjny*> <*plik konfiguracyjny*>
[*długość bufora*] [*liczba buforów*]

Dwa ostatnie parametry wywołania programu *prouter* są opcjonalne. Program zawsze wyświetla wartości przyjętych parametrów procesu komunikacyjnego. Wynikiem wywołania programu *prouter* jest utworzenie *pliku konfiguracyjnego* oraz pliku o nazwie *opis* zawierającego informacje o systemie. *Plik konfiguracyjny* jest plikiem wejściowym dla programu *config* systemu Parallel C. *Wstępny plik konfiguracyjny* różni się od swej ostatecznej postaci kilkoma szczegółami:

- Musi być podany unikatowy identyfikator procesu w komentarzu wiersza definicji procesu połączonego z procesem komunikacyjnym. Komentarze zaczynają się od

znaku wykrzyknika. Brak numeru oznacza, że proces będzie *lokalny*, tzn. że nie jest połączony z procesem komunikacyjnym.

- Definiujemy tylko te kanały, które łączą procesy lokalne.

Program *prouter* kompletuje wstępny plik konfiguracyjny wstawiając procesy komunikacyjne w każdym transputerze oraz tworząc kanały zewnętrzne między nimi. Są ponadto wstawiane kanały wewnętrzne łączące procesy komunikacyjne z procesami użytkownika oraz proces *procIO* służący do obsługi operacji wejścia i wyjścia z każdego transputera.

4.2. Komunikacja z procesem komunikacyjnym

Został opracowany zbiór procedur zebranych w pliku *proc.c* służących do korzystania z procesu komunikacyjnego w procesach użytkownika. Ich definicje są zawarte w pliku *proc.h* i są opisane poniżej.

- funkcja *initRouter*

  void initRouter(CHAN **wej*, CHAN **wyj*);

Funkcja ta służy do przekazania tablicy kanałów wejściowych i tablicy kanałów wyjściowych procesu (parametry funkcji *main*) do modułu *proc*. W ten sposób nie trzeba ich przekazywać przy każdym wywołaniu funkcji pakietu.

- funkcja *wyslij*

  void *wyslij*(int *ile*, void **bufor*, int *dokad*) ;

Funkcja ta służy do przesyłania wiadomości o rozmiarze *ile* bajtów z obszaru pamięci wskazywanego przez *bufor* do procesu o numerze identyfikacyjnym *dokad*.

- funkcja *czytaj*

  void *czytaj*(int **ile*, void **bufor*, int **skad*) ;

Funkcja ta służy do czytania wiadomości do obszaru pamięci wskazywanego przez *bufor*. Funkcja zwraca długość wiadomości w zmiennej wskazywanej przez *ile* oraz numer identyfikacyjny procesu przesyłającego wiadomość w zmiennej wskazywanej przez *skad*.

- funkcja *rozsylanie*

  void *rozsylanie*(int **ile*, void **bufor*) ;

Funkcja ta służy do rozsyłania wiadomości o rozmiarze *ile* bajtów z obszaru pamięci wskazywanego przez wskaźnik *bufor* do wszystkich procesów połączonych z procesem komunikacyjnym.

- funkcja *nrProcesu*

int *nrProcesu*(void) ;

Funkcja ta zwraca identyfikator procesu.

- funkcja *koniecProgramu*

  int *koniecProgramu*(void) ;

Funkcja ta jest potrzebna do zakończenia programu, a właściwie procesu *ProcIO* wykonywanego w procesorze ROOT i służącego do obsługi operacji wejścia i wyjścia.

- funkcja *print*

  void *print*(void) ;

Funkcja ta służy do wyświetlania na ekranie monitora (standardowym wyjściu) łańcucha znaków. Zostanie on poprzedzony komunikatem określającym numer identyfikacyjny procesu, z którego został wysłany.

Ponadto zostały zaimplementowane standardowe funkcje wejścia i wyjścia języka C: *fopen*, *fclose*, *fprintf* i *fscanf* tak, aby były dostępne z każdego transputera.

5. Opis systemu Express

W tym punkcie opiszemy system Express [10], z którym porównane będą czasy przesyłania danych uzyskanych za pomocą zaimplementowanego pakietu komunikacyjnego. Express jest systemem operacyjnym dla sytemów transputerowych. Istnieją wersje tego systemu dostępne dla innych środowisk wieloprocesorowych. Programy mogą być po kompilacji wykonywane na dowolnej maszynie, dla której jest dostępny system Express. System ten składa się z biblioteki dla wybranego języka programowania (korzystaliśmy z biblioteki dla języka Parallel C), z programu ładującego jednorazowo system do transputerów i z narzędzi służących do uruchamiania programów (ang. debugger) i badania ich profilu dynamicznego (ang. profiler). W każdym transputerze może być wykonywany tylko jeden proces. Drogi przesyłania danych między dwoma węzłami są stałe i ustalone w czasie konfiguracji systemu.

Express rozpoznaje 3 rodzaje topologii, dla których istnieje efektywna i bezpieczna metoda przesyłania danych chroniąca przed wzajemną blokadą procesów. Są to kostka, torus i drzewo. Dla pozostałych topologii metoda stosowana w systemie nie chroni przed wzajemną blokadą procesów. Można ustalić liczbę i rozmiar buforów, choć rozmiar buforów nie może być mniejszy od 1Kb. Lista funkcji bibliotecznych dostępnych w syste-

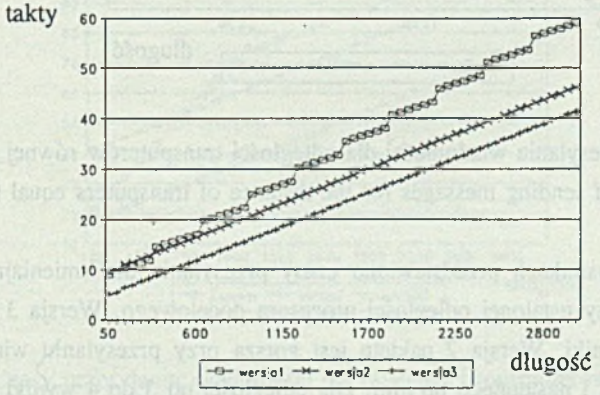
mie Express jest bardzo obszerna, i nie będziemy jej tu omawiać. Podajemy tylko postać wywołań interesujących nas funkcji służących do przesyłania danych:

- funkcja służąca do wysłania wiadomości
  int *exwrite*(char **bufor*, int *długość*, int **procesDocelowy*, int **typInformacji*);
- funkcja służąca do czytania wiadomości
  int *exread*(char **bufor*, int *długość*, int **procesŹródłowy*, int **typInformacji*);

Wiadomości w systemie Express mają przyporządkowany typ zdefiniowany przez użytkownika. Funkcja czytania może zażądać wiadomości danego typu.

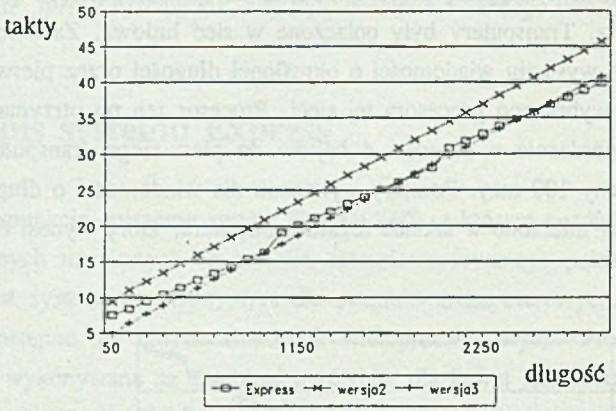
6. Wyniki pomiarów

Badania eksperymentalne, w czasie których porównano zaimplementowany pakiet komunikacyjny z systemem Express, przeprowadzono na 10 transputerach IMS T800 o częstotliwości 20 MHz. Transputery były połączone w sieć liniową. Zadanie programu testowego polegało na wysłaniu wiadomości o określonej długości przez pierwszy procesor sieci liniowej do wybranego procesora tej sieci. Procesor ten po otrzymaniu wiadomości przesyłał potwierdzenie o długości 4 bajtów do pierwszego transputera. Każdy pomiar był wykonywany 100 razy. Pomiary wykonano dla wiadomości o długości od 50 do 3000 bajtów. Czasy mierzono w taktach zegara transputera, który wynosi 64 mikrosekund.



Rys. 1. Czasy przesyłania wiadomości do sąsiedniego transputera dla 3 wersji pakietu
Fig. 1. Timings of sending messages for the distance of transputers equal to 1 for the three versions of the router

Na rys. 1 przedstawiono wyniki porównania trzech wersji zaimplementowanego pakietu przy przesyłaniu wiadomości do sąsiedniego transputera. Wykres dla wersji 1 ma charakter "schodkowy". Jest to spowodowane rosnącą liczbą potwierdzeń, jakie należy wysyłać, gdy liczba pakietów się zwiększa. Wykres dla wersji 2 jest linią prostą, gdyż liczba potwierdzeń nie zmienia się wraz z rosnącą liczbą pakietów. Wersja 3 daje najlepsze wyniki, gdyż nie wysyła się w niej potwierdzeń. Podobne wyniki otrzymano dla innych odległości procesora docelowego. Ponieważ druga wersja pakietu daje zawsze lepsze wyniki niż pierwsza wersja, w dalszym ciągu porównano więc system Express tylko z drugą i trzecią wersją pakietu. W obu pakietach przyjęto rozmiar bufora, dla którego otrzymano najlepsze wyniki. W systemie Express rozmiar ten wynosił 1024 bajtów, a w zaimplementowanym pakiecie 100 bajtów.

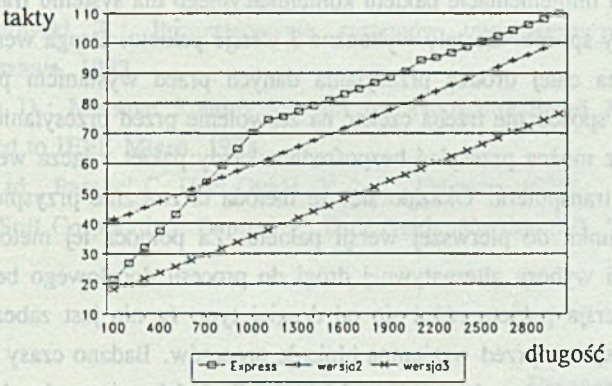


Rys. 2. Czasy przesyłania wiadomości dla odległości transputerów równej 1
Fig. 2. Timings of sending messages for the distance of transputers equal to 1

Na następnych rysunkach przedstawiono czasy przesyłania dla zmieniającej się długości wiadomości przy ustalonej odległości procesora docelowego. Wersja 3 pakietu daje zawsze najlepsze wyniki. Wersja 2 pakietu jest gorsza przy przesyłaniu wiadomości do sąsiedniego procesora i następnego po nim. Dla odległości od 3 do 4 wyniki są podobne, ze zmniejszającą się rożnicą między czasami dla systemu Express, a czasami dla drugiej wersji pakietu wraz z zwiększającą się odległością procesu docelowego. Natomiast zaczynając od odległości 5 wersja 2 pakietu skraca czas przesyłania danych dla większych wiadomości (powyżej 650 bajtów) w stosunku do systemu Express. Można zauważyć, że

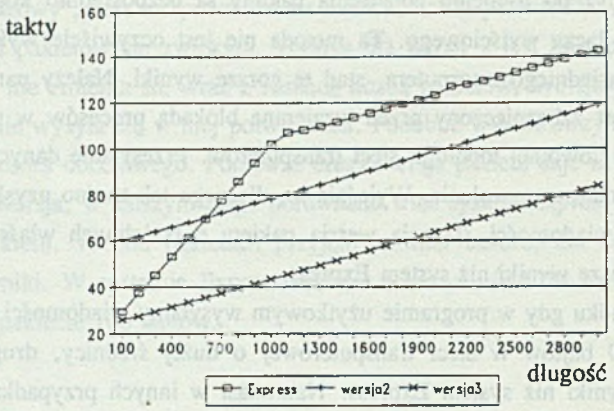
dla wykonanego pakietu zależność czasu od długości przesyłanych danych jest w każdym przypadku liniowa, gdyż po ustaleniu połączenia pakiety są bezpośrednio kopiowane z wejściowego łącza do łącza wyjściowego. Ta metoda nie jest oczywiście wykorzystana przy przesyłaniu do sąsiedniego transputera, stąd te gorsze wyniki. Należy pamiętać, że system Express nie jest zabezpieczony przed wzajemną blokadą procesów w przypadku konfigurowania go dla dowolnej topologii sieci transputerów. Przesyłanie danych odbywa się więc bez oczekiwania na zezwolenie. Wyjaśnia to, dlaczego tak trudno uzyskać lepsze wyniki dla krótszych wiadomości. Trzecia wersja pakietu o podobnych właściwościach daje jednak o wiele lepsze wyniki niż system Express.

Tak więc w przypadku gdy w programie użytkowym wysyłamy wiadomości o długości przekraczającej 650 bajtów w sieci transputerowej o dużej średnicy, druga wersja pakietu daje lepsze wyniki niż system Express. Natomiast w innych przypadkach zaimplementowany pakiet może dać podobne lub gorsze wyniki przy zwiększonym bezpieczeństwie pracy. Przed decyzją o stosowaniu pakietu należałoby więc dla konkretnego programu badać czasy wykonywania programu za pomocą obu pakietów, aby wybrać lepszy z nich. W niektórych programach można oszacować liczbę potrzebnych buforów. W tym przypadku metody zapobiegające wzajemnej blokadzie procesów są niepotrzebne. Należy wtedy korzystać z pierwszej wersji pakietu.



Rys. 3. Czasy przesyłania wiadomości dla odległości transputerów równej 6

Fig. 3. Timings of sending messages for the distance of transputers equal to 6



Rys. 4. Czasy przesyłania wiadomości dla odległości transputerów równej 9
Fig. 4. Timings of sending messages for the distance of transputers equal to 9

7. Podsumowanie

W pracy omówiono implementację pakietu komunikacyjnego dla systemu transputerów połączonych w dowolny sposób. Zostały wykonane 3 wersje pakietu. Druga wersja pakietu rezerwuje bufory na całej drodze przesyłania danych przed wysłaniem pierwszego pakietu danych. W ten sposób nie trzeba czekać na zezwolenie przed przesyłaniem każdego pakietu danych oraz można przesyłać bezpośrednio każdy pakiet z łącza wejściowego do łacza wyjściowego transputera. Okazało się, że metoda ta znacznie przyspiesza przesyłanie danych w stosunku do pierwszej wersji pakietu. Za pomocą tej metody można korzystać z możliwości wyboru alternatywnej drogi do procesu docelowego bez narzutu czasowego. Trzecia wersja pakietu różni się od drugiej tym, że nie jest zabezpieczona, podobnie jak system Express przed wzajemną blokadą procesów. Badano czasy transmisji wiadomości w zależności od długości wiadomości i od odległości procesu docelowego dla trzech wersji pakietu komunikacyjnego oraz dla systemu Express. Najlepsze wyniki otrzymano dla trzeciej wersji pakietu. Jest to wersja, którą należy stosować, gdy niepotrzebna jest ochrona przed wzajemną blokadą procesów, np. gdy wiadomo, jaka będzie maksymalna liczba jednoczesnych komunikatów w systemie. Pierwsza wersja pakietu jest w każdym przypadku gorsza od drugiej wersji. Dla dłuższych wiadomości ($\geq$ 650 bajtów)

druga wersja pakietu jest szybsza od systemu Express dla dłuższych odległości procesu docelowego, a wolniejsza dla odległości 2 i 3.

LITERATURA

[1] Adamo, J.-M., Alhafez Nizar: Minimal, Adaptive and Deadlock-free Routing for Multiprocessors, Laboratoire de l'Informatique du parallélisme, Ecole Normale Supérieure de Lyon, Raport 91-25, July 1991.

[2] Caban D.: Transputer - narzędzie programowania równoległego, Informatyka 6, Warszawa, czerwiec 1988.

[3] Clarke, L., Wilson, G.: Tiny: an efficient routing harness for the Inmos transputer. Concurrency: Practice and experience, Vol. 3, No. 3, (June 1991), 221-245.

[4] Günther, K. D., Prevention of Deadlocks in Packet-Switched Data Transport Systems, IEEE Transactions on communications, Vol. 29, No. 4, April 1981.

[5] Gutkowski R., Szturmowicz M.: Architektura transputera i język OCCAM, Prace IPI PAN, 651, Warszawa, luty 1989.

[6] Hoare C.A.R.: Communicating sequential processes, Communications of the ACM, 21(8), 1978, 666-677.

[7] Kalinowski T.: Programowanie systemów transputerowych, Informatyka 4, Warszawa, 1993.

[8] Talia D.: Message Routing Systems for Transputer-Based Multicomputers, Submitted to IEEE Micro, 1992.

[9] 3L Ltd., Parallel C. User Guide, 3L Ltd., February 1988.

[10] ParaSoft Corporation, Express C. User Guide. Version 3.0, 1990.

Recenzent: Dr Marek Tudruj

Wpłynęło do Redakcji 12 stycznia 1994 r.

Abstract

An implementation of a router for any topologies of transputers is presented. Messages are divided up into packets. There are three versions of the router. Unlike in the first version, the second version reserves the buffers within transputers on the way from the source process to the destination process before sending the first packet of the message. In this version one can choose between the different shortest paths leading to the destination process without additional cost to ensure the correct order of packets receiving. The third version, as opposed to the others, is not deadlock free.

The message transmission times measured in a linear chain of 10 transputers IMS T800 for the three versions of the router and the system Express were compared for different length of messages and different distances. In all cases the third version of the router was the best, and the first version proved to be the worst. The second version of the router is more efficient then Express for messages longer than 650 bytes and distances > 5. In other cases the system Express gives better or similar results than the second version of the router. The second version of the router can be better than the system Express in parallel programs which employ long messages.