ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1993

Seria: INFORMATYKA z.24 Nr kol. 1222

Henryk KRAWCZYK

# OBLICZENIA RÓWNOLEGŁE W SIECI LOKALNEJ

Streszczenie. W pracy zaproponowano środowisko umożliwiające wykonywanie i uruchamianie programów równoległych w sieci LAN. Jego implementacja została wykonana w języku C dla komputerów klasy IBM XT/AT wykorzystujących emulator sieciowy NETBIOS i współpracujących za pomocą połączeń typu Ethernet.

# PARALLEL COMPUTATIONS ON A LOCAL AREA NETWORK

Summary. The paper presents the description of a set of extra functions supporting parallel computations on local area networks. These functions were implemented in C language and can be executed on personal computers (IBM XT/AT) that are equipped with the NETBIOS emulator and that communicate by a Ethernet links.

# PARALLELVERARBEITUNG IM LOKALEN NETZ

Zusammenfassung. Die Arbeit enthält einen Vorschlag des Environments, in dem parallele Programme in LAN-Netze ausgeführt werden können. Das Environment wurde für unter MS-DOS betriebene Rechner, die Netzsystememulator NETBIOS verwenden und mit Ethernet verbindet wurden, in C-Sprache geschrieben.

# 1. Wprowadzenie

W przetwarzaniu równoległym, w przeciwieństwie do przetwarzania sekwencyjnego, próbuje się wykonać jednocześnie kilku kroków algorytmu [7]. To prowadzi do redukcji czasu wykonania całego algorytmu, co tym samym zwiększa efektywność pracy systemu komputerowego [3]. Równoległość może być implementowana na różnych poziomach; od opracowania algorytmu, jego kodowania, do wykonania na odpowiednim systemie komputerowym. Aktualnie, nawet w komputerach osobistych (IBM 586) równoległość jest wprowadzana na etapie wykonywania instrukcji. Poza tym pojawiło się wiele kompilatorów umożliwiających przekształcanie kodu programu sekwencyjnego na wynikowy kod równoległy akceptowalny przez danego typu architekturę równoległą. W przyszłości można oczekiwać powszechnego zastosowania całkowicie nowych, obiektowych i wieloprocesorowych języków programowania, ukierunkowanych na przyjazny zapis i implementację algorytmów równoległych oraz nowych rozwiązań sprzętowych (np. transputery) [4].

W niniejszej pracy przedstawiono koncepcję i podjęto próbę realizacji środowiska, w którym można byłoby w sposób prosty uruchamiać programy równoległe. Przez środowisko należy rozumieć bibliotekę funkcji umożliwiających uzyskanie mechanizmów zapewniających wykonanie algorytmów równoległych zapisanych w języku zrozumiałym dla tego środowiska z minimalną interwencją operatora. Wykonano dwa różne warianty implementacji [2,6] wykorzystujące funkcje Netbiosa [8] oraz sieci D-link. Zebrane doświadczenia są wykorzystywane przy budowie środowiska równoległego w sieci pracującej pod systemem UNIX, gdzie jako węzły sieci wykorzystane są komputery Sun Sparc.

# 2. Podstawowe założenia projektowe

Dotyczą one architektury systemu równoległego, typu przetwarzanych algorytmów oraz sposobu zarządzania przetwarzaniem. Sieć LAN stanowi zbiór systemów komputerowych oraz innych urządzeń do przetwarzania, zapamiętywania, wprowadzania i wyprowadzania informacji w postaci cyfrowej, rozmieszczonych na ograniczonym obszarze, połączonych ze sobą za pomocą środków sprzętowych i programowych w sposób zapewniający szybką wymianę informacji. Najpowszechniejszym ich zastosowaniem jest łączenie między sobą skomputeryzowanych stanowisk pracy. Umożliwia to korzystanie ze wspólnej bazy danych, wymianę poczty elektronicznej, łączenie cząstkowych prac w większe całości. Sieci lokalne pozwalają też zredukować liczbę drogich urządzeń peryferyjnych: drukarek laserowych, ploterów, urządzeń do digitalizacji, pamięci masowych. Gdy pojawią się wymagania na dużą moc obliczeniową, to sieć lokalna posiadajaca pewne dodatkowe mechanizmy współpracy

może być skonfigurowana do systemu równoległego. Stąd istotne staje się przebadanie zdolności funkcjonowania sieci w dwóch trybach pracy: rozproszonym i równoległym [1]. Ponieważ sieci różnią się technologią wykonania (typ procesora, rodzaj połączeń), rodzajami konfiguracji (struktura połączeń, rodzaj dostępu), a także strategią administrowania (funkcje systemu zarzadzającego, dostępne możliwości dla użytkownika), zadanie takie jest niebanalne. Istotne jest też, czy realizowane zadanie zawiera niezależne procedury, które mogą być wykonywane w tym samym czasie. Zakłada się, że ich identyfikacja należy do programisty. Wówczas obliczenia równoległe sprowadzają się do odpowiedniego zarządzania wykonaniem takich niezależnych procedur. Przykładem tego może być wykonywanie operacji na macierzach. Niech A,B i C będą macierzami o tych samych wymiarach. Należy obliczyć macierz A, gdzie A=B*C+C*B+C*C. Sprowadza się to do wykonania operacji mnożenia (*) i dodawania (+). Łatwo zauważyć, że procedury A*B, C*B, C*C mogą być wykonane równolegle. Niewiele jednak problemów daje się tak łatwo podzielić na niezależne procedury. Dlatego też poszukuje się rozwiązań poprzez tworzenie innych struktur powiązań między podproblemami Stosunkowo prosta jest struktura liniowa realizująca przetwarzanie potokowe. Przykładem tego może być sortowanie metodą naprzemienną, która wymaga procesorów połączonych w szereg [5]. W innym przypadku równoległość realizuje się na poziomie zbioru zadań.

System równoległy pracuje na ogół pod wspólnym systemem operacyjnym. Innym rozwiązaniem jest autonomiczna realizacja obliczeń przez procesory, w których każdy korzysta z lokalnego systemu operacyjnego. Przykładem takiego rozproszonego sposobu przetwarzania mogą być sieci komputerowe, gdzie niezależne obliczenia mogą być uruchamiane nawet pod różnymi systemami operacyjnymi. Zaimplementowane algorytmy równoległe powinny być uruchamiane również w takim środowisku. W tym celu trzeba zapewnić najpierw możliwość inicjacji sieci oraz wymaganą konfigurację połączeń między współpracującymi jednostkami. W tak zorganizowanej sieci jednostki obliczeniowe powinny posiadać również możliwość swobodnego przesyłania danych oraz wymiany komunikatów między sobą. W tym przypadku wymiana komunikatów pełni rolę synchronizacji obliczeń równoległych. Poza tym poszczególne jednostki powinny także umieć siebie indentyfikować. Zakłada się, że dana jednostka może komunikować się z każdą, z którą ma zestawione połączenie. W danym momencie jedna jednostka może mieć zestawione połączenie z kilkoma innymi. Tak więc podstawowe cechy środowiska równoległego dla sieci lokalnej są następujące:

- ustalanie konfiguracji połączeń między współpracującymi jednostkami,
- swobodne przesyłanie danych między połączonymi jednostkami,
- wymiana komunikatów między poszczególnymi jednostkami,
- wzajemna identyfikacja jednostek,

- niezależność funkcjonowania jednostek.

Podstawowymi pojęciami używanymi w pracy są jednostka zarządzająca i jednostka robocza. Jednostką zarządzającą będzie ten komputer (węzeł) pracujący w sieci, który służy do komunikacji z operatorem oraz do wprowadzania danych, uruchamiania zadań i wyprowadzania wyników. Można więc stwierdzić, że jest to komputer (host), na którym bezpośrednio pracujemy. Pod względem użytkowym nie różni się on niczym od pojedyńczego komputera pracującego samodzielnie, bez współpracy z innymi jednostkami. W projektowanym środowisku może występować jedna lub więcej jednostek zarządzających. Funkcje usługowe zlecane przez jednostkę zarządzającą będą wykonywały jednostki robocze. W systemie musi być co najmniej jedna taka jednostka. Jednostka robocza przez cały czas wolny, to znaczy kiedy nie wykonuje zadania, czeka na rozkazy od jednostki zarządzającej. Jedyną formą, sposobem aktywacji lub dezaktywacji jednostki roboczej jest odpowiedni rozkaz od jednostki zarządzającej. Współpraca między jednostką zarządzającą a jednostką roboczą nie jest oparta na typowo stosowanym w sieciach komputerowych układzie "server-workstation". Jak wiadomo, w tym układzie server udostępnia swoje zasoby, z których kolejne "workstation" mogą korzystać. Nie mogą one jednak zmusić "server'a" do wykonania czegokolwiek, ani też "server" nie może wpłynąć na zmianę pracy "workstation". Do wytworzenia środowiska równoległego te dodatkowe funkcje są niezbędne. Warto również nadmienić, że pojedyńczy komputer może spełniać obie te funkcje.

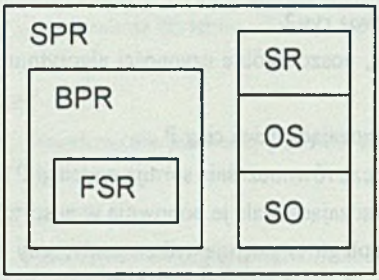
## 3. Podstawowe funkcje środowiska

Realizacja Środowiska Przetwarzania Równoległego (SPR) jest uzależniona od systemu komputerowego, na którym będzie ono zainstalowane, a przede wszystkim od wykorzystywanego programu zarządzającego i przyjętej koncepcji realizacji obliczeń (modelu obliczeń). W tej pracy przyjęto budowę SPR w postaci trzech warstw, aby w przyszłości można byłoby dokonywać zmian w zależności od wykorzystywanego sprzętu i oprogramowania. Warstwy te są następujące:

1. system operacyjny (SO),
2. oprogramowanie sieciowe (OS),
3. środowisko równoległe (SR).

Dodatkową warstwą jest biblioteka programów równoległych (BPR). Układ warstw SPR przedstawia rys. 1. W rozpatrywanym przypadku SO reprezentuje MS-DOS, OS zaś NETBIOS. SR jest zbiorem funkcji dodatkowych, które zostały zaimplementowane do wytworzenia SPR. Zostaną one przedstawione poniżej. BPR jest zaś zbiorem programów użytkowych pracujących w tym środowisku. Aktualnie są to programy (zbiór FSR), które wykorzystano do testowania funkcjonalnego SPR.

Adaptacja sieci lokalnej do środowiska równoległego, oprócz możliwości zmiany konfiguracji i przesyłania danych, wprowadza także możliwość obsługi zadań na różnych jednostkach. Zapewnia to w prosty sposób zorganizowanie wykonywania jednego lub wielu zadań przez jednostki robocze. Wszystkie te mechanizmy są dostępne jako funkcje w języku C. Biblioteka SR składa się z 12 funkcji, które można podzielić na cztery grupy:




Rys. 1. Warstwy projektowanego środowiska. równoległego

Fig. 1. The distinguished layers of parallel environment

a) funkcje ustalania konfiguracji połączeń: należą tutaj tylko dwie funkcje "connect" i "disconnect". Służą one odpowiednio do zestawiania i anulowania połączeń między węzłami sieci,
b) funkcje przesyłania danych: wyróżniono tutaj również dwie funkcje "putdata" oraz "getdata". Odpowiednio służą one do przesyłania i odbierania danych w postaci ciągu znaków,
c) funkcje zlecania zadań: wchodzą tutaj takie funkcje, jak "exectask" (zleca wykonanie zadania), "donetask" (sprawdza, czy zadanie zostało wykonane), "getreslt" (pobiera wyniki zadania), "putreslt" (przesyła wyniki zleconego zadania) i "setinstall" (przesyła dane o zadaniach, które może wykonać),
d) funkcje specjalne: to grupa funkcji realizujacych zadania specjalne, które są niezbędne dla prawidłowego działania pozostałych funkcji. Są to więc funkcje: "sorinit" (inicjacji pracy środowiska), funkcja "sorend" (kończąca pracę środowiska) i funkcja "sorctrl" (realizująca synchroniczne czynności wykonywane przez środowisko).

Funkcje "sorinit" i "sorend" powinny być wykonane tylko raz, pierwsza na początku, druga zaś na końcu programu. Natomiast funkcja "sorctrl" w zależności od potrzeb. Biblioteka funkcji jest wspólna dla jednostki zarządzającej i jednostki roboczej, jednak nie wszystkie funkcje są wykorzystywane przez oba rodzaje jednostek. Jednostka zarządzająca nie korzysta z funkcji "connect" i "setinstall", zaś jednostka robocza "exectask" i "getreslt".

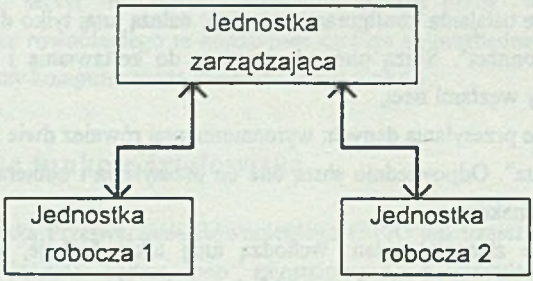
## 4. Przykład realizacji programu równoległego

Jako ilustrację działania SPR rozpatrzmy sortowanie ciągu liczb rzeczywistych, stosując najprostszy algorytm równoległy: Niech będzie dany ciąg P liczb całkowitych. Dzielimy ten ciąg na dwa podciągi P1 i P2. Sortujemy je oddzielnie, a następnie scalamy je w jeden ciąg wynikowy stosując sekwencyjny algorytm sortowania. Konfigurację sieci LAN do wykonania tego algorytmu przedstawia rys.2.

Jak łatwo zauważyć, poszczególne czynności algorytmu sortowania można podzielić na następujące kroki:

- jednostka zarządzająca dzieli ciąg P,
- jednostki robocze równocześnie sortują podciągi P1 i P2 ,
- jednostka zarządzająca scala je ponownie w posortowany ciąg P'.

Przetwarzanie równoległe występuje tylko wtedy, gdy obie jednostki robocze pracują jednocześnie. Użytkownik wykorzystujący SPR do sortowania ciągu liczb powinien przygotować odpowiednie programy sortujące. Poniżej przedstawiono opisy działania programu jednostki zarządzającej i jednostek roboczych.




Rys. 2. Przyjęta konfiguracja sieci dla sortowania równoległego

Fig. 2. The assumed network confiuration for parallel sorting

**Jednostki zarządzająca:**

```
#include <fsr.h>                /* dołączenie prototypów FPR          */
#define SIZE ???                /* maksymalna ilość liczb do sortowania */
#define BYTESIZE SIZE*8         /* rozmiar pamięci zajmowanej przez liczby  */
/* program sortuje liczby w tablicy Table */
void main( )
{
    double Table[ SIZE ] ;
    int        size = 10, zad1, zad2;
    uint    dlug;
```

```
/* zainicjowanie pracy stanowiska, pomocnicze zbiory będą na dysku D:\, jednostka będzie
miała nazwę master */
    sorinit( "D:\\", "master" ) ;
/* program chodzi w pętli; loop symbolizuje pętlę. W celu uruchomienia programu należy
wstawić tu jeden z rodzajów pętli - while, for, do; */
    loop
/* funkcja wczytująca liczby do posortowania do tablicy Table, size jest rozmiarem tablicy, to
znaczy określa ilość liczb */
    WczytajLiczby(Table, size ) ;
/* zlecenie wykonania zadania, posortowania pierwszej połówki tablicy Table przez wolną
jednostkę roboczą - slave */
    zad1 = ZdalneSort( Table, size/2-1 ) ;
/* zlecenie wykonania zadania, posortowania drugiej połówki tablicy Table przez wolnego
slave */
    zad2 = ZdalneSort( Table+size/2, size-size/2+1 ) ;
/* oczekiwanie na zakończenie wykonania zadań, to jest posortowania obu połówek. Funkcja
sorctrl obsługuje przesłania danych */
    do {
        sorctrl( ) ;
    } while( !donetask( zad1 ) && !donetask( zad2 ) ) ;
    dlug = BYTESIZE ;
/* odebranie wyników posortowania pierwszej połówki tablicy
    getreslt( zad1, (char *)Table, &dlug ) ;
/* odebranie wyników posortowania drugiej połówki tablicy */
    getreslt( zad2, (char *)Table+size/2, &dlug ) ;
/* scalenie obu posortowanych połówek w uporządkowany ciąg liczb */
    split( Table, 0, size/2, size ) ;
    endloop ;
/* zakończenie pracy */
    sorend( ) ;
}/*main*/
int ZdalneSort( tab, roz )
double tab[ ] ;
int     roz ;
{
    static char bufor[ BYTESIZE+8 ] ;
```

```
/* pierwsze 8 bajtów w przesyłanych danych niech będzie bajtami informacyjnymi. Wpiszemy
tu ilość liczb do posortowania przez slave */
    sprintf( bufor, "%02d%06c", roz, '0' ) ;
/* przepisanie liczb do bufora, który zostanie przesłany do slave */
    memcpy( bufor+8, tab, sizeof(double)*(roz+1) ) ;
/* właściwe zlecenie wykonania zadania sort */
    return( exectask( "sort", bufor, dlug+8 ) ) ;
}/*ZdalneSort*/
```

**Jednostka robocza:**

Programy dla jednostki roboczej 1 i jednostki roboczej 2 różnią się tylko nazwą nadawaną przy inicjalizacji.

```
#include <fsr.h>    /* dołączenie prototypów FSR    */
void main( )
{
/* zainicjowanie pracy stanowiska, pomocnicze zbiory będą na dysku D:\, jednostka będzie
miała nazwę slave Nr x, x 1 lub 2 */
    sorinit( "D:\\", "slave?" ) ;
/* zestawienie połączenia z jednostką o nazwie "nazwa". W tym miejscu zamiast "nazwa"
należy podać nazwę, jaka została nadana master'owi. */
    connect( nazwa ) ;
/* zainstalowanie się jako wykonawcy zadań o nazwie "sort" */
    setinstall( "sort" ) ;
/* program wykonuje się w pętli; loop symbolizuje pętlę. W celu uruchomienia programu
należy wstawić zamiast loop jeden z rodzajów pętli - while, for, do; */
    loop
/* oczekiwanie na zgłoszenia wykonania zadań o nazwie sort. Funkcja sorctrl obsługuje
przesłania danych */
    sorctrl( ) ;
    endloop ;
}/*main*/
/* funkcja, która jest niezbędna dla umożliwienia obsługi wykonania zgłoszonego zadania. W
programie "slave" musi znaleźć się funkcja remotecall. Funkcja ta zostaje wywołana przez
SPR. Numer "nr" jest kolejnym numerem nazwy podanej dla funkcji setinstall (numery są od
zera). W naszym przypadku nazwa "sort" ma numer 0. Wywołanie z innym numerem
powinno więc zostać zignorowane */
void remotecall( nr, buff, len )
```

```
int nr ;
char  buff[ ] ;
unsigned len ;
{
   if( nr == 0 )  sortowanie( buff, len ) ;
}/*remotecall*/
void sortowanie(par,len)    /* wykonanie sortowania nadesłanych liczb */
char  par[];
uint  len;
{
   char  buf[ 8 ] ;
   int x ;
   unsigned roz ;
/* odczytanie ilości liczb do posortowania sekwencyjnego */
   sscanf( par, "%d", &x ) ;
/* skopiowanie liczb do obszaru wskazanego przez Table. Liczby, zgodnie z przyjętym
założeniem, są zapisane począwszy od ósmego bajtu przesyłanych danych. */
   memcpy( Table, par+8, roz=(x+1)*sizeof(double) ) ;
/* wykonanie sortowania */
   quicksort( 0, x ) ;
/* przesłanie wyników */
   putreslt( (char *)Table, roz ) ;
}/*sortowanie*/
```

## 5. Uwagi końcowe

Zaimplementowane środowisko zostało przetestowane dla wybranych algorytmów numerycznych i kombinatorycznych [6]. Dokonano również pomiarów czasu ich wykonania. Dla przykładu przedstawionego w rozdziale 4 uzyskane wyniki zaprezentowano w tabeli 1, gdzie czas wykonania wyrażono w sekundach, a dokładność pomiaru wynosi 0.05 s. Jak wynika z tabeli 1, wraz ze wzrostem liczby komputerów czas sortowania maleje, gdy liczba jednostek roboczych nie przekracza 4, a następnie wzrasta. Fakt ten wynika ze stosunkowo niewielkiej ilości danych (1000 elementów ciągu), co powoduje, że krótszy czas sortowania nie rekompensuje strat związanych z przesłaniem danych. Poza tym dla rosnącej liczby procesorów wzrasta ilość przesłań, co powoduje wzrost czasu scalania bardziej rozproszonego ciągu. Opracowane środowisko wykorzystuje się na zajęciach dydaktycznych z "Systemów równoległych".

Tabela 1

Czas sortowania w funkcji liczby jednostek roboczych

| Liczba jednostek roboczych | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| Czas sortowania równoległego | 7.41 | 6.53 | 5.49 | 5.88 | 6.38 |

## LITERATURA


[1] G.Bernard, A. Duda: Primitives for Distributed Computing in a Heterogeneous Local Area Network Environment, IEEE Trans. Software Eng., No. 12, 1989.

[2] P. Brudło, H. Krawczyk: Przetwarzanie rozproszone w komputerowych sieciach lokalnych. Materiały Krajowego Sympozjum Telekomunikacji, KST"93, Bydgoszcz 1993.

[3] W. Iszkowski: Przyśpieszanie obliczeń przez ich rozpraszanie, Informatyka 7, 1989.

[4] H. Krawczyk: Architektura i oprogramowanie mikrokomputerowych systemów równoległych. Materiały XIII Krajowej Konferencji Teoria Obwodów i Układy Elektroniczne, Bielsko-Biała 1990.

[5] W.S. Luk, F. Ling: An Analytic/Empirical Study of Distributed Sorting on a Local Area Network, IEEE Trans. on Software Eng. No. 5, 1989.

[6] A. Stec: Programowanie równoległe w oparciu o sieć Ethernet, Praca dyplomowa, 1991.

[7] M.M. Sysło: Maszyny i algorytmy równoległe, Informatyka 11-12, 1988, 1, 1989.

[8] A. Tucholski: Netbios - zasada działania i sposób użytkowania, Informatyka 11-12, 1988, 1, 1989.



Recenzent: Dr hab. inż. Stanisław Kozielski
Prof. Politechniki Śląskiej

Wpłynęło do redakcji dnia 15 września 1993



## Abstract

The paper analyses the possibility of parallel computation on local area networks. The considered network consists of 8 nodes (PC XT/AT clones) that are connected by Ethernet links and working under LANSMART system. It is required to define at least one master

node for coordination and at least one slave-node for processing. To support parallel computation three layers of software are used. They are shown in Fig. 1 and represent operation system (SO) network communication software (OS) and parallel environment (SR), respectively. Additionally a library of parallel programs (BPR) can be utilised which also contains programs (FSR) used for testing the parallel network configuration. Previously, our implementation based on MS-DOS and NETBIOS. Presently we try adopted our parallel environment for networks working under UNIX.

Twelve functions of the SR layer are defined and implemented in C language. They can be divided into four groups as follows:

1. functions establishing logical connections between two nodes; such structures as stars, chains, meshes can be created,
2. functions supporting data transmissions between two nodes; from input file to output file,
3. functions initiating task executions and handling obtained results,
4. functions creating and deleting work of parallel environment in the LAN,

The above functions can be performed in the network according to the program prepared by a user. Their execution do not require operator interventions. To illustrate possibilities of the proposed parallel environment, the example of parallel sorting is included into the paper. We restrict our analysis to the configuration presented in Fig. 2. Behaviour of the master and slaves nodes is described in detail in section 4. The computation time of the algorithm for 1000 elements of data sequences as a function of the number of computations nodes is given in Table 1. It shows, that a balance between computation time and communication time should be achieved in the network to perform parallel computation in efficient way.