ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996

Seria: INFORMATYKA z. 31 Nr kol. 1348

Katarzyna HARĘŻLAK
Piotr BAJERSKI

# REALIZACJA SYSTEMU STEROWANEGO PRZEPŁYWEM ARGUMENTÓW PRZY UŻYCIU SYSTEMU LINDA

Streszczenie. W pracy zaprezentowano różne możliwości implementacji Systemu Sterowanego Przepływem Argumentów z wykorzystaniem systemu LINDA. W artykule tym dla prezentacji poszczególnych rozwiązań wykorzystano problem sortowania danych. Artykuł ten zawiera także krótką charakterystykę systemu LINDA.

## IMPLEMENTATION OF THE ARGUMENT FLOW DRIVEN SYSTEM USING THE LINDA SYSTEM

Summary. In the paper different implementation methods of the Argument Flow Driven System using the Linda System are presented. The different implementation methods are exemplified by the data sorting problem. The paper includes a short description of the Linda System.

Pracę wykonano w ramach grantu KBN nr 3 P406 011 04.

## REALISIERUNG EINES DURCH DEN ARGUMENTENFLUß GESTEUERTEN SYSTEMS UNTER VERWENDUNG VOM LINDA-SYSTEM

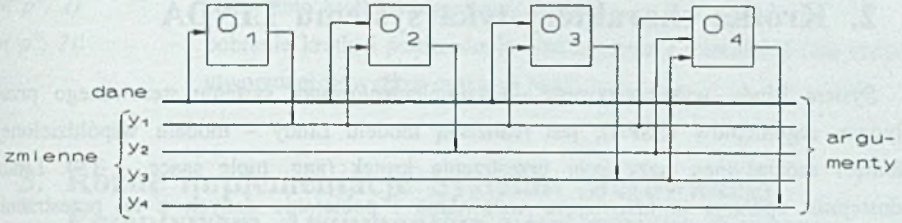
Zusammenfassung. Im Artikel wurden unterschiedliche Möglichkeiten der Implementierung eines durch den Argumentenfluß gesteuerten Systems unter Verwendung vom LINDA-System präsentiert. Für die Darstellung einzelner Lösungen wurde das Problem der Datensortierung ausgenutzt. Der Artikel enthält auch eine kurze Charakteristik des LINDA-Systems.

# 1. Wstęp

Obecnie jednym z najczęściej rozpatrywanych problemów w informatyce jest próba przyspieszenia wykonywania różnego typu obliczeń i operacji. Niewątpliwy wpływ na ten rozwój sytuacji ma coraz większy zasięg i wykorzystanie sprzętu komputerowego w naszym codziennym życiu. Istnieją dwa kierunki rozwiązywania tego problemu. Pierwszy z nich obejmuje doskonalenie rozwiązań sprzętowych, tworzenie coraz szybszych procesorów, pamięci i dysków o coraz krótszych czasach dostępu. Drugi dotyczy sfery oprogramowania, w ramach której tworzone są nowe języki i systemy pozwalające na tworzenie programów, których wydzielone części mogą wykonywać się równolegle. Narzędzia te pozwalają z kolei na realizację różnych idei przetwarzania współbieżnego.

Artykuł ten zawiera analizę możliwości wykorzystania jednego z grupy wcześniej wspomnianych systemów – systemu Linda – do realizacji Systemu Sterowanego Przepływem Argumentów (SSPA).

W SSPA zakłada się, że zadania rozwiązywane przy pomocy komputerów można podzielić na niezależne operacje wykonywane równolegle z wykorzystaniem różnych procesorów lub stacji sieci. Warunkiem wykonania danej operacji jest pojawienie się wszystkich niezbędnych argumentów na odpowiadających im szynach, którymi połączone są operacje. Rysunek 1 przedstawia przykładowy schemat Systemu Sterowanego Przepływem Argumentów. Na rysunku tym szyny argumentów podzielone zostały na dwie części: dane będące danymi wejściowymi dla systemu oraz zmienne reprezentujące wyniki pośrednie i końcowe. Dane zadanie można zrealizować przy pomocy SSAP, jeśli algorytm rozwiązania tego zadania da się przedstawić w postaci kanonicznej [1] oraz można określić macierz zmiennych formy kanonicznej [1] przedstawiającą zależności pomiędzy rezultatami uzyskiwanymi w wyniku realizacji poszczególnych operacji systemu. Jeśli macierz ta jest macierzą trójkątną górną, bez elementów na przekątnej głównej, to określa ona bezpośrednio strukturę powiązań jednostek opera-




Rys. 1. System Sterowany Przepływem Argumentów realizujący algorytm (1)
Fig. 1. A System controlled by flow of arguments and executing algorithm (1)

cyjnych SSPA rozwiązującego dany problem. System przedstawiony na rys. 1 realizuje algorytm o następującej postaci kanonicznej:

$$\begin{aligned} y_1 &= O_1(a_1) \\ y_2 &= O_2(y_1, a_2) \\ y_3 &= O_3(y_1, a_3) \\ y_4 &= O_4(y_1, y_2, y_3) \end{aligned} \tag{1}$$

gdzie: $a_1$, $a_2$, $a_3$ - dane,
$y_1$, $y_2$, $y_3$, $y_4$ - zmienne,

a jego struktura oparta została na macierzy zmiennych formy kanonicznej zamieszczonej poniżej.

| | $y_1$ | $y_2$ | $y_3$ | $y_4$ |
|---|---|---|---|---|
| $y_1$ | | × | × | × |
| $y_2$ | | | | × |
| $y_3$ | | | | × |
| $y_4$ | | | | |

(2)

W dalszej części artykułu zawarto analizę różnych możliwości implementacji SSPA w systemie Linda.

## 2. Krótka charakterystyka systemu LINDA

System Linda, wykorzystywany do zaimplementowania systemu sterowanego przepływem argumentów (SSPA), jest realizacją modelu Lindy – modelu współdzielonej pamięci asocjacyjnej, nazywanej przestrzenią krotek (ang. tuple space – TS). Linda udostępnia operacje umożliwiające wstawianie i pobieranie danych z tej przestrzeni. Dane te, nazywane krotkami (ang. tuples), składają się z ciągów pól.

Do zrealizowania obliczeń równoległych potrzebne są dwa języki: język pozwalający zapisać rozwiązywany problem oraz język umożliwiający tworzenie nowych procesów i ich koordynację. Pierwszym z nich jest język C, drugim język C–Linda będący implementacją modelu Lindy. Można powiedzieć, że język C–Linda jest zanurzony w języku C, dodając do niego operacje wstawiania i pobierania krotek do/z przestrzeni krotek.

Operacja **in** próbuje pobrać z przestrzeni krotek (ang. tuple space – TS) krotkę, szukając wśród znajdujących się w niej krotek takiej, która pasuje do wzorca, podanego jako argument tej operacji. Dopasowanie zachodzi, gdy: liczba pól w krotce i we wzorcu jest taka sama, istnieje zgodność typów odpowiadających sobie pól oraz wartości znajdujące się we wzorcu równają się wartościom w krotce. Jeżeli nie znaleziono pasującej krotki, wykonanie operacji jest blokowane do czasu jej pojawienia. Gdy wzorzec można dopasować do kilku krotek, jedna z nich jest wybierana niedeterministycznie. Pasująca krotka jest usuwana z przestrzeni krotek i za zmienne we wzorcu są podstawiane wartości pól krotki. Operacja **rd** działa analogicznie do operacji **in**, z tym wyjątkiem że odczytana krotka nie jest usuwana z przestrzeni krotek. Operacja **out** umieszcza w TS krotkę będącą jej argumentem. Jeżeli którekolwiek pole w operacji **out** zawiera wyrażenie, jest ono obliczane przed wstawieniem krotki do przestrzeni krotek.

Operacja **eval** służy do tworzenia nowych procesów na zdalnych komputerach. Dla każdego pola zawierającego wywołanie funkcji tworzony jest na jednym z komputerów biorących udział w obliczeniach proces wartościujący tę funkcję. Procesy te są niezależne i wykonują się równolegle. Operacja **eval** kończy się po utworzeniu procesów wartościujących.

Przykłady operacji języka C-Linda:

*int i;*

*eval("funct", f())* – utworzenie nowego procesu, wykonującego funkcję *f()*,

*out("p", 2)* – wstawienie krotki do przestrzeni krotek,

*in("p", 2)* – pobranie krotki z przestrzeni krotek,

*rd("p", 2)* – odczytanie krotki z przestrzeni krotek,

*in("p", ?i)* – pobranie krotki i podstawienie pod zmienną *i* wartości 2 (dla krotki utworzonej powyższą operacją **out**).

## 3. Różne implementacje Systemu Sterowanego Przepływem Argumentów przy użyciu systemu LINDA

Rozważania dotyczące realizacji SSPA zostaną oparte na możliwości implementacji operacji i szyny SSPA w modelu Lindy przez odpowiednio proces Lindy i krotkę. Przez proces Lindy będzie rozumiany proces wyznaczający wartość wskazanej funkcji programu, stworzony przez Lindę na którymś z dostępnych komputerów i wykonujący się równolegle względem innych procesów Lindy. W każdym z rozwiązań poza procesami realizującymi operacje SSPA istnieje proces koordynujący całość pracy. Jest on nazywany nadzorcą.

Najprostszym rozwiązaniem jest zdefiniowanie funkcji realizujących operacje SSPA, w których pobranie danych z szyn danych realizowane jest operacjami **rd**, argumentów z szyn argumentów operacjami **in**, a wyprowadzanie wyników operacjami **out**. W ten sposób przesył argumentów między częściami programu jest w nim zapisany "na sztywno". Na początku wykonania programu proces nadzorcy tworzy osobny proces Lindy dla każdej funkcji (będącej implementacją operacji SSPA) przy użyciu operacji **eval** i przygotowuje dane, wystawiając je na szyny danych. Następnie czeka na pojawienie się rezultatu oznaczającego koniec obliczeń. Proces nadzorcy kończy wtedy procesy Lindy i wyprowadza wynik. Głównym problemem takiego rozwiązania jest wymaganie, aby każdej operacji SSPA odpowiadał fizycznie jeden procesor (plus jeden procesor dla nadzorcy). Wymaganie to powinno być spełnione, jeśli chcemy osiągnąć maksymalne przyspieszenie. Jednak, nawet jeśli uda się dostarczyć tyle procesorów, ich moc obliczeniowa jest marnowana, gdyż dany procesor wykonuje tylko jedną przypisaną mu operację, a resztę czasu spędza bezczynnie.

Wymaganie użycia dużej liczby komputerów można obejść, przyjmując, że na jednym procesorze może być uruchomiony więcej niż jeden proces Lindy. Prowadzi to jednak do sytuacji, w której pewne węzły będą przeciążone, a inne bezczynne, jeżeli operacje wykonujące się w tym samym czasie zostaną ulokowane na tych samych procesorach. Trudno zapobiec takiej sytuacji, gdyż Linda nie dostarcza mechanizmów przydziału procesu na zadany procesor.

Rodzi się tutaj pytanie, czy procesy dla wszystkich operacji SSPA muszą być inicjowane na początku wykonania programu. Innym problemem jest taka implementacja systemu, aby zminimalizować czas wykonania programu i być może osiągnąć maksymalną efektywność wykorzystania sprzętu. Poniższe rozważania podejmują próbę analizy tego problemu.

**Pierwszy wariant SSPA**

Rozwiązanie to nakłada ciężar organizacji pracy całego sytemu na proces nadzorcy. Jego zadaniem jest uruchomienie wszystkich wykonawców realizujących funkcje systemu, wprowadzenie danych do przestrzeni krotek oraz obsługa wyników pośrednich i końcowych. Implementacja ta zakłada, że w danej chwili wykonywane są tylko te operacje, dla których są dostępne argumenty. Przydzielanie operacji na wolne procesory można wykonać uruchamiając nowy proces, jak tylko któryś z uruchomionych wykonawców zakończy swoje działanie, zwalniając procesor. Na procesorze tym uruchamiana jest operacja, dla której argumenty są skompletowane. Proces nadzorcy musi w takich rozwiązaniach nieustannie nadzorować stan obliczeń określając, które argumenty są dostępne oraz które operacje są gotowe. Kierowanie realizacją poszczególnych funkcji systemu z poziomu koordynatora nie daje jednak możliwości wyboru wierzchołka sieci do realizacji konkretnych funkcji.

**Drugi wariant SSPA**

Implementację SSPA możemy również otrzymać, wykorzystując uniwersalnych wykonawców potrafiących zrealizować dowolną operację. Na każdym z procesorów jest uruchamiany jeden wykonawca. W rozwiązaniu tym krotka danych musi nieść ze sobą informację o operacji, którą na danych tych należy wykonać. Wymaga to wzbogacenia struktury krotki o pole, zawierające kod operacji. Każdy z wykonawców pobiera z TS krotkę z danymi, wykonuje wskazaną operację (wywołując lokalnie odpowiadającą jej funkcję) i zwraca wynik do TS. W implementacji tej eliminowane są opóźnienia (w stosunku do poprzedniego wariantu) przy tworzeniu operacji – gdzie za każdym razem trzeba tworzyć nowy proces. Ze względu na specyfikę operacji Lindy (operacje blokujące, niedeterministyczne pobieranie krotek) wymagane jest, aby wszystkie szyny dochodzące do tej samej operacji były opisane przez jedną krotkę. Program wykonawcy uniwersalnego ma tę dodatkową zaletę, że może wybierać funkcję, która będzie wykonywana przez aktualnie przydzielony komputer. Osiąga się to przez testowanie wierzchołka, na którym uruchomiony został wykonawca i – w zależności od wyniku testu – wykonanie przeznaczonej dla niego operacji. Oznacza to, że pomimo braku w systemie Linda mechanizmów przydzielania zadań na poszczególne wierzchołki, istnieje możliwość wykonania złożonych operacji przez mocniejsze komputery.

**Trzeci wariant SSPA**

Innym wariantem jest stworzenie w TS krotek niezależnie opisujących operacje SSPA i szyn (każdej operacji i każdej szynie danych odpowiada jedna krotka). W rozwiązaniu tym uniwersalni wykonawcy pobierają operacje i sprawdzają, czy na dochodzących do niej szynach są dostępne argumenty. Każda szyna argumentów ma swój adres. Krotki operacji, oprócz swojego identyfikatora i kodu ją reprezentującego, zawierają adresy szyn z danymi dla tej operacji. Organizując pętle pobierania kolejnych krotek opisujących operacje, należy uwzględnić wspomnianą wyżej cechę systemu Linda niederministycznego pobierania krotek z TS. Jeśli zawartość krotek operacji ograniczona byłaby do kodu operacji i adresu szyn danych związanych z daną operacją, istniałoby niebezpieczeństwo, że wykonawca będzie pobierał i zwracał jedną i tę samą krotkę (testował tę samą operację), w przypadku gdy dane dla operacji nie będą gotowe. Jedną z metod eliminacji tego problemu jest wprowadzenie do struktury krotki pola wskazującego gotowość danych dla konkretnej operacji. Dzięki temu wykonawca będzie przeglądał kolejne operacje, wykonując te, dla których argumenty zostały już obliczone. Również i w tej implementacji, ze względu na wykorzystanie uniwersalnych wykonawców, można sterować wykonaniem operacji przez konkretne wierzchołki.

Kolejny rozdział zawiera analizę możliwości wykorzystania różnych wariantów SSPA w zadaniu sortowania danych.

## 4. System Sterowany Przepływem Argumentów realizujący operacje sortowania

Analiza możliwości implementacji różnych typów SSPA w systemie oparta zostanie na problemie sortowania danych, które podzielone na podciągi porządkowane są we wzajemnie równoległych procesach. Tak posortowane ciągi podlegają łączeniu, aż do uporządkowania wszystkich danych. Poniżej zamieszczona została postać kanoniczna algorytmu sortowania danych podzielonych na cztery podciągi. Operacje $O_1$, $O_2$, $O_3$ i $O_4$ reprezentują sortowanie pojedynczego podciągu (odpowiednio $d_1$, $d_2$, $d_3$, $d_4$), natomiast operacje $O_5$, $O_6$ i $O_7$ odpowiadają operacjom łączenia posortowanych podciągów.

$$\begin{aligned} y_1 &= O_1(d_1) \\ y_2 &= O_2(d_2) \\ y_3 &= O_3(d_3) \\ y_4 &= O_4(d_4) \\ y_5 &= O_5(y_1, y_2) \end{aligned} \tag{3}$$

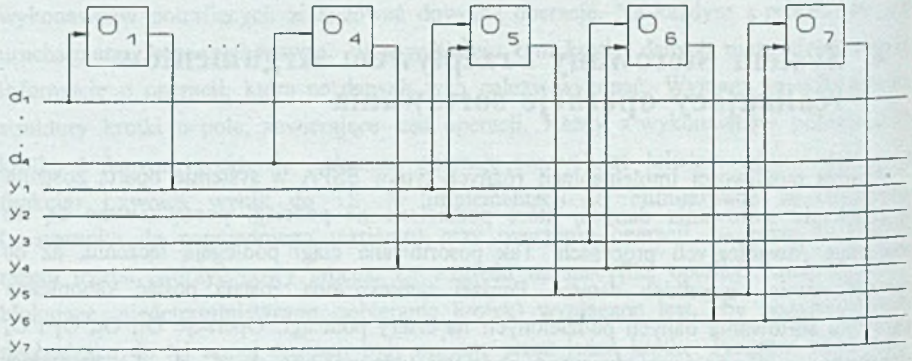
$$y_6 = O_6(y_3, y_4)$$
$$y_7 = O_7(y_5, y_6)$$

Macierz zmiennych dla tego algorytmu ma następującą postać:

| | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ |
|---|---|---|---|---|---|---|---|
| $y_1$ | | | | | × | | |
| $y_2$ | | | | | × | | |
| $y_3$ | | | | | | × | |
| $y_4$ | | | | | | × | |
| $y_5$ | | | | | | | × |
| $y_6$ | | | | | | | × |
| $y_7$ | | | | | | | |

(4)

Z macierzy tej wynika, że operacje sortowania oraz operacje łączenia ($O_5$, $O_6$) mogą wykonywać się równolegle. Prowadzi ona do struktury SSPA przedstawionej na rys. 2.




Rys. 2. System Sterowany Przepływem Argumentów realizujący algorytm (3)
Fig. 2. A System controlled by flow of arguments and executing algorithm (3)

Opierając się na rozważaniach zawartych w pracy [5] można określić – na podstawie macierzy zmiennych formy kanonicznej – przyspieszenie S realizacji algorytmu sortowania dla czterech podciągów. Zgodnie z [5] wynosi ono S = 7/3.

SSPA może być oczywiście realizowany dla dowolnej liczby podciągów. Zwiększa się wtedy liczba operacji, które mogą być realizowane równolegle.

Dalsza część rozdziału obejmuje omówienie problemu sortowania realizowanego przez różne warianty SSPA.

**Wariant 1**

W pierwszym z rozważanych rozwiązań całością działań kieruje proces nadzorcy. Jego zadaniem jest przygotowanie krotek zawierających przeznaczone do sortowania ciągi oraz synchronizacja wykonywania poszczególnych operacji. Wykonanie zadania będzie wymagało wykorzystania dwóch funkcji realizujących odpowiednio operacje sortowania i łączenia. W tym wariancie SSPA na początku programu uruchamiane zostaną tylko te procesy, które będą zajmowały się sortowaniem podciągów. Wykorzystywana jest w tym celu funkcja **eval**("worker",sortuj()). W kolejnym kroku nadzorca wprowadza dane do przestrzeni krotek w formacie ("dana", ciąg, flaga), gdzie "dana" jest identyfikatorem krotki, ciąg - zawiera dane do sortowania, flaga - określa, czy ciąg jest posortowany. Wykonanie postawionego zadania kontrolowane jest przez proces nadzorcy, który sprawdza, czy w przestrzeni krotek znalazły się wszystkie krotki z uporządkowanymi danymi. Oznacza to, że wykonawcy zakończyli sortowanie zwalniając procesory. Nadzorca uaktywnia kolejnych wykonawców (**eval**("worker", łącz())), których zadaniem jest połączenie posortowanych danych w jeden uporządkowany ciąg wynikowy. Ilość etapów, w ramach których będą odbywały się operacje łączenia, zależna jest od przyjętej na początku liczby podciągów. Liczba ta, licznik pobranych krotek oraz długość podciągu, jest różna dla każdego poziomu łączenia i uaktualniana przez proces nadzorcy po uporządkowaniu wszystkich par na danym etapie. Zakończenie łączenia na danym poziomie nie pociąga za sobą zakończenia pracy wykonawców. Następuje to dopiero po osiągnięciu ostatniego poziomu. Poniżej zamieszczono fragmenty programu zapisanego w pseudokodzie mogącego realizować SSPA zgodnie z wariantem 1.

```
main()
{
 deklaracja zmiennych;
 wykonuj w pętli /* inicjalizacja wykonawców */
   { eval("worker", sortuj()); }
 wykonuj w pętli /* wprowadzenie danych do TS */
    { out("dana", ciąg, flaga); }
 wykonuj w pętli /* odczyt krotek posortowanych, flaga = 1 */
    { in("dana", ?ciąg, 1); }
 ....
 wykonuj w pętli /* uruchomienie kolejnych wykonawców */
    { eval("worker_1",lacz(j)); }
 wykonuj w pętli
```

```
    {
    wprowadzenie krotek zapewniających synchronizację łączenia do TS;
    wykonuj w pętli /* wprowadzenie krotek do łączenia */
        { out("dana", ciąg, flaga); }
    wykonuj w pętli /* odczyt połączonych krotek, flaga = 1 */
        { in("dana", ?ciąg, 1); }
    }
}

sortuj()
{
 pobierz krotkę z TS;
 sortuj;
 wyprowadź wynik do TS
}

łącz()
{
 pobierz krotki do łączenia;
 połącz;
 wyprowadź wynik do TS
}
```

**Wariant 2**

W drugiej implementacji, jak wspomniano w rozdziale 3, zakłada się istnienie uniwersalnych wykonawców potrafiących wykonać dowolną, z zaprojektowanych w systemie SSPA, operację. W tym przypadku są to sortowanie i łączenie. Oprócz uruchomienia wykonawców rola procesu nadzorcy ogranicza się tu do wprowadzenia krotek z danymi do przestrzeni krotek oraz pobrania wynikowej krotki z uporządkowanym ciągiem danych. Pozostałe pośrednie wyniki przekazywane są pomiędzy poszczególnymi wykonawcami bez udziału procesu nadzorcy. Rozwiązanie to pociąga za sobą rozszerzenie informacji zawartej w krotce danych. Przybiera ona następującą postać:

("dana", kod operacji, ciąg, długość ciągu)

Wykonawca, pobierając krotkę, testuje kod operacji i wywołuje odpowiednią lokalną funkcję. Ponieważ krotki pobierane są z przestrzeni krotek w sposób niedeterministyczny, może zdarzyć się próba łączenia krotek o różnej długości. W celu wyeliminowania tego problemu wprowadzone zostało pole "długość ciągu", które jest elementem krotki zapewniającym synchronizację łączenia.

W rozwiązaniu tym można natknąć się na sytuację wspomnianą w poprzednim rozdziale, a mianowicie kilkakrotną próbę pobrania tej samej krotki. Ponieważ łączenie ma przebiegać na dwóch podciągach, w przestrzeni krotek muszą istnieć dwie krotki o tym samym rozmiarze, aby operacja ta mogła zakończyć się sukcesem. Ze względu na niedeterministyczne pobieranie krotek z TS mogą one być kilkakrotnie pobierane przez tych samych wykonawców, zanim dojdzie do ich połączenia.

Aby tego uniknąć, można wzbogacić krotkę o element będący jej numerem oraz o element wskazujący kolejność pobierania krotek do łączenia. W pierwszej kolejności przez proces wykonawcy pobierane są krotki do łączenia o parzystym numerze, czego wskaźnikiem jest flaga ustawiana np. na 0. Do łączenia z daną krotką pobierana jest krotka z numerem nieparzystym większym o 1 i flagą odpowiednio ustawioną na 1. Krotka danych ma wtedy postać:

("dana", nr krotki, kod operacji, ciąg, długość ciągu, flaga gotowości)

Poniżej, podobnie jak w dla wariantu 1, zamieszczono fragmenty pseudokodu mogącego realizować wariant 2.

```
main()
{
 deklaracje zmiennych;
 wykonuj w pętli /* inicjalizacja wykonawców */
   { eval("worker", sortuj()); }
 pobranie krotki z wynikiem końcowym;
}


worker()
{
 wykonuj w pętli
   {
   /* pobranie krotki z TS */
    in("dana", 0 , ?nr, ?kod, ?ciąg, ?dł_ciągu, ?flaga_got);
    /* wybór funkcji w zależności od kodu operacji */
    switch (kod)
      {
      case 0: sortuj(ciąg);
      case 1: lacz(ciąg);
      case 2: break;  /* zakończenie łączenia */
      }
   }
}
```

```
sortuj(ciąg)
{
 sortuj ciąg;
 wprowadź wynik do TS;
}

łącz(ciąg)
{
 pobierz krotkę nr+1;
 połącz krotki;
 wprowadź wynik do TS;
}
```

**Wariant 3**

W ostatnim wariancie proces nadzorcy przygotowuje wszystkie krotki, które będą niezbędne dla rozwiązania problemu, na początku wykonania programu. Oprócz krotek danych pojawiają się również krotki operacji, których liczba zależna jest od deklarowanej liczby podciągów. Wykonawcy pozostają również uniwersalni, z tą różnicą że z przestrzeni krotek pobierają krotki operacji, a nie danych, jak to było w rozwiązaniu poprzednim. Postać takich krotek jest następująca:

("operacja", kod, adrwe, adrwy, flaga gotowości)

Zawierają one adresy szyn, na których znajdują się dla nich argumenty (adrwe), oraz adresy szyn, na które należy zwrócić wynik (adrwy). Warunkiem pobrania operacji do wykonania jest ustawienie flagi gotowości na 1. Operacja jest gotowa, jeśli na szynach danych są wszystkie argumenty niezbędne do jej wykonania. W przypadku sortowania krotka z danymi zawiera następujące elementy:

("dana", adrwe, ciąg, długość ciągu),

a dla łączenia

("dana", adrwe, ciąg1, ciąg2, długość ciągu)

We wszystkich krotkach operacji sortowania flaga gotowości ustawiana jest przez proces nadzorcy po wprowadzeniu do TS podzielonych na podciągi danych. Dla operacji łączenia pole to jest uzupełniane przez wykonawcę, który jako drugi wypełnia krotkę danych. Wykonawcy kończą swoją pracę, gdy pobrane zostaną wszystkie krotki operacji, co w efekcie kończy sortowanie danych.

```
main()
{
 deklaracje zmiennych;
 wykonuj w pętli /* inicjalizacja wykonawców */
  { eval("worker", worker()); }
 wykonuj w pętli /* wprowadź krotki dla sortowania */
  { out ("dana", adrwe, ciąg, dł_ciągu) }
 wykonuj w pętli /* wprowadź krotki dla łączenia */
  { out ("dana", adrwe, ciąg1, ciąg2, dł_ciągu) }
 wykonuj w pętli /* wprowadź krotki operacji */
  { out ("op, kod, nr_op, adrwe, adrwy, flaga_got) }
 pobierz krotkę wynikową;
}

worker()
{
 wykonuj w pętli
  {
   /* pobranie krotki operacji z TS */
   in("op",?kod,?nr,?adrwe,?adrwy,1);
   /* wybór funkcji w zależności od kodu operacji */
    switch(kod)
      {
       case 0: sortuj(adrwe,adrwy);
       case 1: lacz(adrwe,adrwy);
       case 2: break; /* zakończenie łączenia */
      }
  }
}

sortuj(adrwe,adrwy)
{
 /* pobierz krotkę o adresie adrwe */
 in("dana", adrwe, ?ciąg, ?dł_ciągu);
 sortuj;
 /* pobierz krotkę o adresie adrwy; */
 in("dana",adrwy, ?ciąg1, ?ciąg2, dł_ciągu);
 jeśli ciąg1 jest pusty
   umieść wynik sortowania w polu ciąg1;
 w przeciwnym wypadku
   {
   umieść wynik sortowania w polu ciąg2
   /* pobierz krotkę operacji, dla której przygotowano ciągi */
   in("op",kod, adrwy, adrwy, 0);
   ustaw flagę na 1;
   /* wprowadź krotkę operacji do TS */
    out("op", kod, adrwy, adrwy, 1);
   }
}

łacz(adrwe,adrwy)
```

```
{
 /* pobierz krotkę o adresie adrwe; */
 in("dana", adrwe, ?ciąg1, ?ciąg2, ?dł_ciągu);
 połącz;
 /* pobierz krotkę o adresi adrwy; */
 in("dana",adrwy, ?ciąg1, ?ciąg2, dł_ciągu);
 jeśli ciąg1 jest pusty
   umieść wynik łączenia w polu ciąg1
 w przeciwnym wypadku
   {
   umieść wynik łączenia w polu ciąg2
   /* pobierz krotkę operacji, dla której przygotowano ciągi */
   in("op",kod, adrwy, adrwy, 0);
   ustaw flagę na 1;
   /* wprowadź krotkę operacji do TS */
    out("op", kod, adrwy, adrwy, 1);
   }
}
```

## 5. Wnioski

Jak wynika to z przedstawionych w artykule rozważań, system LINDA jest bardzo elastycznym narzędziem organizującym równoległe przetwarzanie. Jego dużą zaletą jest wykorzystanie współdzielonej pamięci asocjacyjnej, która daje możliwość bezpośredniego dostępu do danych wszystkim działającym współbieżnie procesom. Istnienie wspólnej przestrzeni krotek pociąga za sobą łatwość implementacji Systemu Sterowanego Przepływem Argumentów.

Zamieszczone przykłady różnych wersji systemu realizującego sortowanie danych dowodzą, jak dużą swobodę pozostawia system LINDA w organizacji SSPA. Rozwiązanie danego problemu może być zrealizowane na kilka sposobów, o wyborze którego decyduje twórca systemu, kierując się różnymi względami (np. wygodą implementacji, efektywnością czasową).

Otwarty pozostaje problem ustalenia poziomu szczegółowości, na którym będzie realizowany system. Postawić sobie należy pytanie, czy istnieje możliwość realizacji takiego SSPA, który umożliwi rozwiązywanie szerokiej klasy zadań dotyczących danej dziedziny. Ważne jest, by jeden system mógł zapewnić rozwiązania nie jednemu, lecz większej liczbie zagadnień o wspólnej tematyce. Nie bez znaczenia jest też efektywność czasowa takiego systemu. Istotne jest więc znalezienie tzw. "złotego środka" między szybkością pracy a poziomem szczegółowości rozwiązania. Jest to problem bardzo interesujący i pozostaje przedmiotem ciągłych badań.

## LITERATURA

[1] Węgrzyn S.: Systemy sterowane przepływem operacji i systemy sterowane przepływem argumentów. Zeszyty Naukowe Politechniki Śl., Seria Informatyka, z. 24, 1993.

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

[3] C-Linda User's Guide & Reference Manual, Scientific Computing Associates Inc. 1993.

[4] Carriero N., Gelernter D.: How to write parallel programs, A first course. The MIT Press 1990.

[5] Węgrzyn S.: Przyspieszenie realizacji algorytmów w systemach sterowanych przepływem argumentów. Zeszyty Naukowe Politechniki Śl., Seria Informatyka, 1994.

Recenzent: Dr inż. Jerzy Grzywocz

Wpłynęło do Redakcji 5 kwietnia 1996 r.

## Abstract

The paper presents different ways of implementation of Argument Flow Driven System (AFDS) using the Linda System. Figure number 1 shows a schema of an exemplary AFDS, consisting of some operational units and data links joining them. Equations (1) and (2) show for this exemplary AFDS the canonical form and the variable matrix of the canonical form respectively. The chapter 2 briefly describes the Linda System. The following chapters contains a theoretical analysis of using the Linda System for implementation of AFDS.

There are shown three different implementations of AFDS. In all of them operational units correspond to processors, the AFDS operations are created using as templates functions, and data lines are implemented as tuples. The first implementation assumes that

the master process deals with associating AFDS operations with their operands and assigning the ready operations to processors.

The second implementation uses universal workers able to perform any operation of the AFDS. On each of the processors used only one worker runs. In this solution a tuple contains the operation code and all the data needed to perform the operation.

The last implementation is based on creation in the tuple space tuples describing AFDS operations and data links (each operation and each data links is related to one tuple). In this solution universal workers take operations from the tuple space and check if the needed operands are available.