ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1993

Seria: INFORMATYKA z. 24 Nr kol. 1222

Stanisław KOZIELSKI
Bogumił BIENIOK

ZASTOSOWANIE SIECI KOMPUTEROWEJ DO SYMULACJI KOMPUTERA PRZEPŁYWOWEGO REALIZUJĄCEGO ZADANIA WYSZUKIWANIA DANYCH

Streszczenie. W artykule przedstawiono symulator komputera sterowanego przepływem danych oparty na sieci stacji roboczych Sun. Symulator ten zastosowano do równoległej realizacji zadań wyszukiwania danych.

APPLYING OF COMPUTER NETWORK TO SIMULATION OF DATAFLOW COMPUTER IN DATA RETRIEVING PROBLEMS

Summary. A simulator of a dataflow computer based on a network of Sun workstations is presented in the paper. Simulator was applied to parallel realization of data retrieving problems.

DIE AUSNUTZUNG DES COMPUTER-NETZWERKES ZUR SIMULATION DES DATENFLUß-COMPUTERS, DER DIE DATENAUSSUCHAUFGABEN REALISIERT

Zusammenfasssung. Im Artikel wurde der Simulator des Datenfluß-Computers vorgestellt, der auf Basis des Netzwerkes der SUN-Workstations realisiert wurde.

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

Dieser Simulator wurde zur parallelen Ausführung der Datenaussuchaufgaben verwendet.

1. Wstęp

W pracy przedstawiono symulator komputera przepływowego zastosowany do wykonywania zadań wyszukiwania, oparty na sieci stacji roboczych.

W pierwszej części artykułu przedstawiono wybrany sposób zapisu zadań wyszukiwania oparty na algebrze relacji. Następnie rozpatrzono możliwość rozpraszania obliczeń niezbędnych do wykonania takich zadań. Przedyskutowano problemy związane ze sterowaniem współbieżną realizacją rozproszonych obliczeń. Analizując różne możliwości takiego sterowania, zwrócono uwagę na analogię między drzewem obliczeń dla wyrażeń algebry relacji a grafem przepływu danych.

W drugiej części artykułu - nawiązując do przykładowej realizacji komputera przepływowego - przedstawiono wykonany symulator tego komputera. Omówiono też narzędzia programowe wykorzystane do tej realizacji.

Ostatnia część prezentuje rezultaty kilku eksperymentów przeprowadzonych z wykorzystaniem symulatora.

2. Przyjęty sposób zapisu zadań wyszukiwania danych

W pracy będziemy rozważali problem wyszukiwania danych opierając się na relacyjnym modelu danych.

Dla tego modelu dane są organizowane w dwuwymiarowe tablice (relacje), których nagłówki, nazywane schematami, są zbiorami nazw atrybutów. Wiersze tablic (krotki relacji) zawierają wartości atrybutów. Na poziomie fizycznej organizacji danych tablicom odpowiadają pliki danych, wierszom tablic - rekordy, zaś nazwom kolumn - nazwy pól.

Istnieje szereg języków umożliwiających formułowanie pytań kierowanych do relacyjnych baz danych. W niniejszej pracy ograniczymy się do języków poziomu algebry relacji [9, 10, 11].

Najważniejszymi operacjami dla języków tego poziomu są:

selekcja: tworząca w wyniku nową tablicę składającą się z tych wierszy tablicy wejściowej, które spełniają zadany warunek selekcji,

projekcja: tworząca nową tablicę ze wskazanych kolumn tablicy wyjściowej,

złączenie: tworzące jedną tablicę z dwóch tablic wyjściowych; wiersze nowej tablicy powstają ze złożenia tych wierszy obu tablic, które spełniają warunek złączenia (np. równość artybutów wspólnych w obu tablicach).

W celu zilustrowania sposobu zapisu pytań za pomocą wymienionych operatorów rozpatrzymy bazę danych składającą się z następujących relacji (plików danych):

Studenci (nazwisko, imię, średnia, album, akademik, adres, semestr)
Stypendia (album, rok, miesiąc, kwota)
Oceny_zal (album, przedmiot, ocena_z, semestr_z)
Oceny_egz (album, przedmiot, termin, ocena_e, semestr_e)
Przedmioty (przedmiot, nazwa)
Języki (album, język, stopień)
Wyjazdy (album, kraj, cel, okres).

Wspólnym atrybutem w sześciu plikach jest "album" oznaczający numer indeksu studenta i jednoznacznie go identyfikujący. Wspólnym atrybutem w trzech plikach jest "przedmiot" oznaczający kod przedmiotu. W pozostałych przypadkach nazwy atrybutów określają znaczenie gromadzonych danych.

Dla tak zdefiniowanej bazy rozważmy następujące pytanie:

Znaleźć nazwiska i imiona studentów, którzy mieszkają w akademiku, uzyskali średnią ocen większą od 3.3, ocenę z egzaminu z matematyki w pierwszym terminie większą lub równą 3.5 i w miesiącu październiku otrzymali stypendium socjalne

Zapis tego pytania za pomocą operatorów algebry relacji w najbardziej ogólnej postaci można przedstawić następująco:

$$\text{wynik} := \pi_X\,(\sigma_{w1 \wedge w2 \wedge w3 \wedge w4}(\text{studenci} \bowtie \text{stypendia} \bowtie \text{przedmioty} \bowtie \text{oceny\_egz}))$$

gdzie X : {nazwisko, imię}

w_1 : (akademik < > ' ') ∧ (średnia > 3.3)

w_2 : (stypendium = 'S') ∧ (miesiąc = '10')

w_3 : nazwa = 'MATEMATYKA'

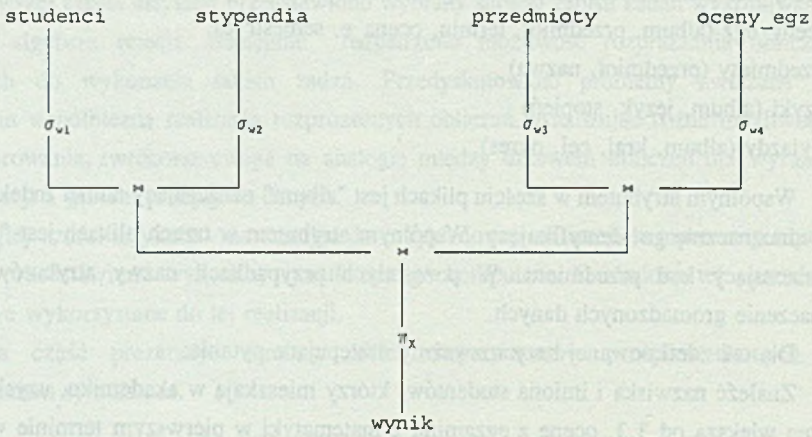
w_4 : (ocena_e > = 3.5) ∧ (termin = 1)

Wykorzystując reguły optymalizacji wyrażeń algebry relacji [10] można przekształcić ten zapis do postaci:

$$\text{wynik} := \pi_X\,(\sigma_{w1}(\text{studenci}) \bowtie \sigma_{w2}(\text{stypendia}) \bowtie \sigma_{w3}(\text{przedmioty}) \bowtie \sigma_{w4}(\text{oceny\_egz}))$$

Powyższy zapis może być podstawą wygenerowania kilku wariantów programu wyszukiwania w konkretnym języku zapytań bazującym na algebrze relacji. Warianty te mogą się przede wszystkim różnić kolejnością wykonywania operacji selekcji (wraz z ewentualnymi projekcjami) i operacji złączeń.

Dla dalszych rozważań przedstawimy powyższe wyrażenie w formie drzewa obliczeń (rys. 1), o specjalnie wybranej symetrycznej postaci. Taka struktura umożliwia rozważanie równoległej realizacji wymaganych w tym przykładzie obliczeń.



Rys. 1. Przykładowe drzewo obliczeń dla rozważanego zadania wyszukiwania
Fig. 1. Example of computation tree for considered data retrieving problem

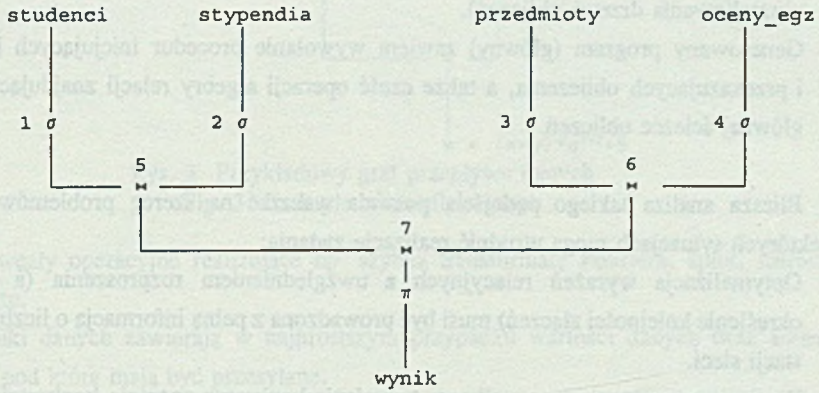
W przypadku, kiedy program taki jest realizowany w typowym jednoprocesorowym komputerze, kolejność wykonywania operacji selekcji nie ma znaczenia. Kolejność wykonywania operacji złączeń może mieć znaczenie, ale problem optymalnej kolejności wykonywania tych operacji w komputerze jednoprocesorowym nie będzie dyskutowany [10, 11]. Zwróćmy w tym miejscu uwagę na fakt, że zazwyczaj w programowej realizacji następującej po sobie operacje selekcji i projekcji, a także złączenia i projekcji są wykonywane łącznie, co istotnie zmniejsza czas realizacji tych operacji.

3. Problem rozpraszania obliczeń w sieci komputerowej

Rozważać będziemy sytuację, kiedy przedstawione poprzednio zadanie będzie realizowane w lokalnej sieci komputerowej. Przyjmijmy, że w sieci pracuje szereg stacji,

które w danej chwili są wolne lub też których moc obliczeniowa może być (chociażby częściowo) wykorzystana do realizacji wyróżnionego zadania.

W takim przypadku celowe byłoby rozważenie możliwości równoległej realizacji pewnych fragmentów obliczeń. Na rys. 2 wyodrębniono w prezentowanym uprzednio drzewie obliczeń poszczególne etapy realizacji rozważanego zadania wyszukiwania.



Rys. 2. Wyodrębnienie fragmentów obliczeń do realizacji równoległej
Fig. 2. Distinction of computation fragments to parallel realization

Zauważmy, że dysponując 4 stacjami sieci moglibyśmy rozmieścić operacje selekcji o numerach 1...4 na tych stacjach, inicjując ich równoległą realizację. Po wykonaniu tych obliczeń na wybranych dwóch stacjach mogłyby być realizowane złączenia 5 i 6. Ostatnie złączenie wraz z projekcją 7 mogłoby być zrealizowane w dowolnej stacji prawdopodobnie zaś w tej, w której oczekiwane są rezultaty obliczeń.

Po tak zarysowanej propozycji rozproszenia obliczeń należy rozważyć sposób sterowania wykonaniem zadań wyszukiwania.

3.1. Klasyczny model sterowania współbieżną realizacją rozproszenia obliczeń

Przez model klasyczny będziemy rozumieli podejście oparte na następujących założeniach:

- W rozważanej sieci komputerowej jest dostępna biblioteka procedur (programów) realizujących operacje algebry relacji. Programy te mogą zostać uruchomione w wolnych stacjach sieci, pozostając w stanie oczekiwania na zlecenie im zadań.

- Dysponujemy narzędziami programowymi umożliwiającymi przekazywanie komunikatów do odległych stacji, inicjowanie programów w takich stacjach i synchronizowanie wykonania programów w wielu stacjach [12].
- W procesie interpretacji pytania zapisanego jako wyrażenie algebry relacji etap optymalizacji takiego wyrażenia prowadzony jest z uwzględnieniem możliwości rozproszenia obliczeń w określonej liczbie stacji sieci (co prowadzi do odpowiedniego ukształtowania drzewa obliczeń).
- Generowany program (główny) zawiera wywołanie procedur inicjujących inne stacje i przekazujących obliczenia, a także część operacji algebry relacji znajdujących się na głównej ścieżce obliczeń.

Bliższa analiza takiego podejścia pozwala wskazać na szereg problemów, które w niektórych sytuacjach mogą utrudnić realizację zadania:

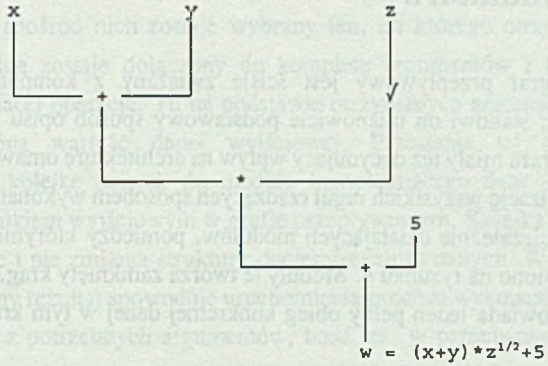
- Optymalizacja wyrażeń relacyjnych z uwzględnieniem rozproszenia (a dokładniej, określenie kolejności złączeń) musi być prowadzona z pełną informacją o liczbie wolnych stacji sieci.
- Wygenerowany program wynikowy przypisuje konkretne operacje konkretnym stacjom sieci.
- Niemożność zrealizowania operacji we wskazanej stacji (wskutek np. wyłączenia stacji czy też zajęcia jej przez nieświadomego użytkownika) praktycznie uniemożliwia dokończenie programu. Należałoby w takim przypadku powtórzyć generację programu zmieniając (być może automatycznie) wykaz aktualnie wolnych stacji sieci.

Omówione problemy mogą znaleźć rozwiązanie - przynajmniej częściowe - w przedstawionym dalej innym modelu sterowania realizacją obliczeń.

3.2. Sterowanie współbieżną realizacją obliczeń przy wykorzystaniu modelu przepływowego

Punktem wyjścia do rozważenia takiego modelu jest analogia między drzewem obliczeń tworzonym dla zadań wyszukiwania a grafem przepływu danych.

Graf przepływu danych [7, 8] zawiera węzły operacyjne i decyzyjne, węzły łączące i rozgałęziające oraz krawędzie, po których przenoszone są tzw. znaczniki danych (rys.3). Najistotniejsze w rozważanej analogii węzły operacyjne mogą realizować różne rodzaje operacji arytmetycznych i innych. Dla pierwszych modeli komputerów przepływowych były to operacje elementarne, odpowiadające rozkazom maszynowym, tzn. dodawanie, odejmowanie, mnożenie itd. Komputery dla bardziej specjalizowanych zastosowań [1, 3]



Rys. 3. Przykładowy graf przepływu danych
Fig. 3. Example of dataflow graph

zawierają węzły operacyjne realizujące np. szybką transformatę Fouriera, splot, filtrację sygnałów itp.

Znaczniki danych zawierają w najprostszym przypadku wartości danych oraz adresy docelowe, pod które mają być przesyłane.

Kluczowym dla koncepcji komputera przepływowego założeniem jest uzależnienie wykonania operacji w węźle od obecności znaczników danych na wszystkich krawędziach wejściowych do danego węzła.

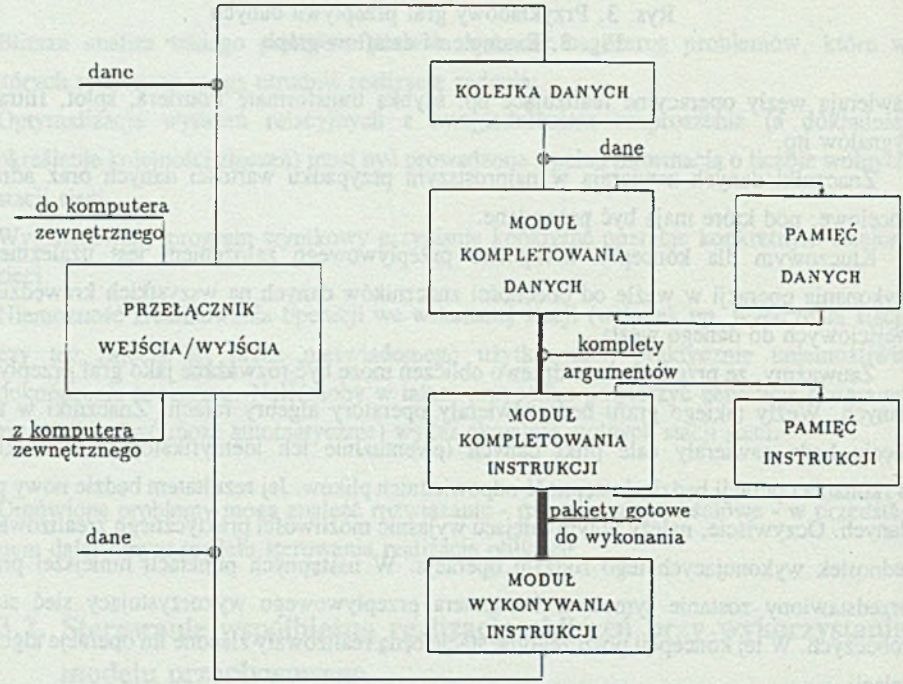
Zauważmy, że przedstawione drzewo obliczeń może być rozważane jako graf przepływu danych. Węzły takiego grafu będą zawierały operatory algebry relacji. Znaczniki w tym ujęciu będą zawierały całe pliki danych (ewentualnie ich identyfikatory). Warunkiem wykonania operacji będzie dostępność odpowiednich plików. Jej rezultatem będzie nowy plik danych. Oczywiście, należy w tym miejscu wyjaśnić możliwości praktycznego zrealizowania jednostek wykonujących tego rodzaju operacje. W następnych punktach niniejszej pracy przedstawiony zostanie symulator komputera przepływowego wykorzystujący sieć stacji roboczych. W tej koncepcji poszczególne stacje będą realizowały zlecone im operacje algebry relacji.

Rozważmy teraz (uprzedzając opis samego symulatora) ewentualne korzyści wynikające z przyjęcia takiego modelu sterowania współbieżną realizacją obliczeń:

- Opis zadania obliczeniowego w konwencji grafu przepływu danych nie wymaga żadnych dodatkowych działań w celu wskazania fragmentów programu realizowanych równolegle.
- Po wyborze gotowych do realizacji operacji są one kierowane do modułu wykonania dysponującego dowolnym (niepustym) zbiorem jednostek przetwarzających, czyli stacji sieci. Tak więc analiza i optymalizacja wyrażeń algebry relacji nie jest uwarunkowana znajomością aktualnej dyspozycyjności stacji sieci.

4. Opis symulatora

Przedstawiony graf przepływowy jest ściśle związany z komputerem sterowanym przepływem danych: stanowi on mianowicie podstawowy sposób opisu programu dla tego komputera. Cechy grafu miały też decydujący wpływ na architekturę omawianego komputera, która umożliwia realizację wszystkich reguł rządzących sposobem wykonania instrukcji grafu. Składa się ona z 4 niezależnie działających modułów, pomiędzy którymi przepływają dane tak, jak to przedstawiono na rysunku 4. Moduły te tworzą zamknięty krąg. Wykonaniu jednej instrukcji grafu odpowiada jeden pełny obieg konkretnej danej w tym kręgu.



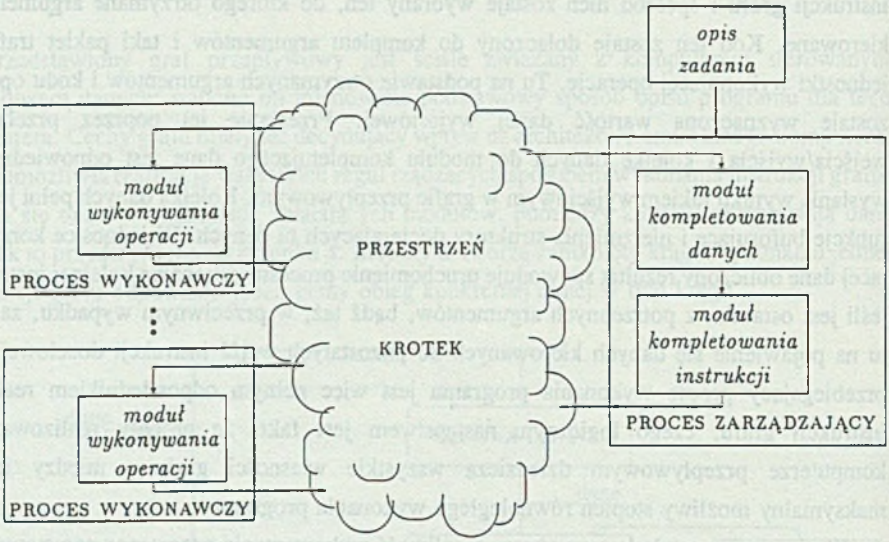
Rys. 4. Organizacja przykładowego komputera przepływowego (wg [6])
Fig. 4. Example of dataflow computer organization [6]

Szczegółowy opis budowy i działania komputera sterowanego przepływem danych można znaleźć w [6], warto jednak zwrócić uwagę na fakt, że realizacja programu zawartego w module kompletowania instrukcji odpowiada realizacji operacji grafu przepływowego. Pojawieniu się argumentów na łukach wejściowych operacji w grafie, a tym samym umożliwieniu wykonania tej operacji, w komputerze przepływowym odpowiada skompletowanie tych argumentów przez moduł łączący dane. Tak utworzony komplet danych

przesyłany jest do modułu kompletującego instrukcje. Tu znajdują się kody wszystkich instrukcji grafu i spośród nich zostaje wybrany ten, do którego otrzymane argumenty są kierowane. Kod ten zostaje dołączony do kompletu argumentów i taki pakiet trafia do jednostki wykonującej operacje. Tu na podstawie otrzymanych argumentów i kodu operacji zostaje wyznaczona wartość danej wyjściowej. Przesłanie jej poprzez przełącznik wejścia/wyjścia i kolejkę danych do modułu kompletującego dane jest odpowiednikiem wysłania wyniku łukiem wyjściowym w grafie przepływowym. Kolejka danych pełni jedynie funkcje buforujące i nie zmienia struktury docierających tu danych. W jednostce kompletującej dane obliczony rezultat spowoduje uruchomienie procesu wykonania kolejnej instrukcji, jeśli jest ostatnim z potrzebnych argumentów, bądź też, w przeciwnym wypadku, zaczeka tu na pojawienie się danych kierowanych do pozostałych wejść instrukcji docelowej. Tak przebiegający proces wykonania programu jest więc pełnym odpowiednikiem realizacji instrukcji grafu, czego logicznym następstwem jest fakt, że procesy realizowane w komputerze przepływowym dziedziczą wszystkie własności grafu - między innymi maksymalny możliwy stopień równoległego wykonania programu.

W niniejszym artykule rozważymy możliwość wykorzystania programowego symulatora takiej właśnie architektury do współbieżnej realizacji zadań wyszukiwania. Zamiast jednak, jak to zostało przyjęte przez tworców komputera przepływowego, wykorzystania rozkazów wykonujących operacje elementarne (proste operacje arytmetyczne, rozkazy sterujące przebiegiem procesu), w przedstawionym symulatorze jako operacje atomowe przyjęto bardziej złożone operacje, charakterystyczne dla klasy zadań, do których rozwiązywania symulator został przygotowany. W ten sposób w pełni korzystając z automatycznej synchronizacji procesów można uniknąć zbyt dużego wpływu nakładu obliczeniowego związanego z symulacją na efektywność programu. W prezentowanym symulatorze stworzono możliwość dosyć prostego definiowania takich niedekomponowalnych operacji i przystosowywania tym samym programu do rozwiązywania nowej klasy zadań. Cała praca programisty nad uwspółbieżnieniem problemu, dla którego znane jest rozwiazanie sekwencyjne, polegać więc będzie na opisie problemu za pomocą grafu przepływu danych. Będzie się to wiązało z wydzieleniem niezależnych fragmentów programu, które następnie zostaną uznane za operacje atomowe w przedstawianym symulatorze i w miarę możliwości sprzętowych zostaną wykonane współbieżnie.

Rozpatrując budowę programu symulatora na najwyższym poziomie ogólności (rys. 5) można wyróżnić dwa procesy: proces zarządzający kojarzeniem znaczników i instrukcji programu oraz proces wykonujący operacje określone przez instrukcje programu. Pierwszy proces zostaje uruchomiony tylko w jednym egzemplarzu, procesy wykonawcze mogą być natomiast uruchomione w kilku (co najmniej w jednym) i to na różnych stacjach roboczych. W ten sposób użytkownik ma wpływ na uzyskany poziom równoległości zadania: więcej



Rys. 5. Ogólna struktura symulatora
Fig. 5. General view of simulator

procesów wykonawczych oznacza większą zdolność do równoległej realizacji obliczeń, która będzie wykorzystana, jeśli w wykonywanym zadaniu wystąpi odpowiednio duża liczba operacji realizowalnych współbieżnie.

Do realizacji komunikacji między procesami (przekazywanie danych i pakietów gotowych do wykonania) wykorzystano system POSYBL, będący ogólnodostępną wersją systemu Linda. Podstawowym pojęciem używanym w tym systemie jest *przestrzeń krotek*, przy czym *krotka* rozumiana jest jako sekwencja pól o określonym typie, co odpowiada pojęciu rekordu w wielu językach programowania. POSYBL dostarcza operatorów umożliwiających umieszczanie krotek w ich zbiorze (przestrzeni), wyjmowanie ich stamtąd, a także odczytywanie wartości ich pól bez usuwania samych krotek z przestrzeni. Dokładny opis tego systemu można znaleźć w [4, 5].

Proces zarządzający umieszcza krotki zawierające pakiety opisujące operacje gotowe do wykonania w przestrzeni krotek systemu POSYBL. Pierwszy wolny proces wykonujący zadania pobiera ten pakiet, realizuje opisane w nim operacje i rezultat swej działalności, w postaci pakietu wynikowego, umieszcza z powrotem w przestrzeni krotek. Stamtąd jest on pobierany przez proces zarządzający, który na jego podstawie formułuje nowe zadania do wykonania. W ten sposób już nawet na tym najogólniejszym poziomie przejawia się

przepływowy sposób realizacji programu, ta charakterystyczna reakcja łańcuchowa: wykonanie operacji pociąga za sobą realizację wszystkich operacji od niej uzależnionych.

Proces zarządzający realizuje funkcje modułu kompletowania danych i jednostki kompletującej instrukcje w komputerze przepływowym. Realizacja funkcji jednostki wykonującej instrukcje została powierzona procesowi wykonawczemu. Funkcja kolejki danych jest natomiast realizowana przez system POSYBL, który przechowuje pakiet w przestrzeni krotek tak długo, jak to jest konieczne.

Zanim rozpocznie się równoległe rozwiązywanie zadania, proces zarządzający odczytuje zawartość pliku opisującego zadanie, które należy rozwiązać, a także zawierającego początkowe dane, które inicjują proces obliczeniowy. Po zakończeniu tego etapu wstępnego proces zarządzający rozpoczyna sterowanie realizacją zadania. Samo wykonanie zadania przebiega w sposób identyczny ze sposobem działania komputera sterowanego przepływem danych. Jeśli dana (stanowiąca zawsze argument instrukcji) przybywa do procesu zarządzającego, podejmowana jest próba znalezienia pozostałych argumentów tej instrukcji, do której jest ona kierowana. W przypadku powodzenia tak utworzony zestaw argumentów łączony jest z kodem instrukcji określającym operację, jaka ma zostać na nich wykonana. W ten sposób powstaje pakiet gotowy do wykonania, który opuszcza proces zarządzający i jest umieszczany w przestrzeni krotek. Stamtąd trafia do procesu wykonawczego, który uruchamia odpowiedni proces obliczeniowy konieczny do wykonania nakazanej operacji i utworzenia danych wynikowych. Te za pośrednictwem przestrzeni krotek trafiają z powrotem do procesu zarządzającego.

Najbardziej zasadnicza różnica w stosunku do pierwszych rozwiązań komputerów przepływowych występuje w module wykonywania instrukcji. Realizowane tu instrukcje nie są, jak to już zostało zasygnalizowane, prostymi operacjami arytmetycznymi czy logicznymi. Są to programy (lub raczej biblioteki procedur lub funkcji) o dowolnie dużej złożoności, które zostały wyznaczone podczas przygotowywania systemu do wykonywania danej klasy zadań. W module wykonawczym następuje więc nie tyle wykonanie operacji, co raczej wywołanie znajdującej się tu funkcji. Kod operacji można więc traktować jako numer funkcji, którą należy wywołać z podanymi argumentami, by otrzymać rezultat.

Wyjaśnienia wymaga jeszcze plik opisujący zadanie, odczytywany przez proces zarządzający. Powstaje on w wyniku działania kompilatora, który przetwarza program opisujący graf przepływowy do postaci zrozumiałej dla procesu zarządzającego.

Do zapisywania zadań zaprojektowany został specjalny język, wzorowany na języku TASS używanym do programowania komputera przepływowego [6]. Umożliwia on opis grafu przepływowego, dlatego przed przystąpieniem do zapisywania programu pomocne jest narysowanie odpowiedniego grafu. Tworzenie programu polega wtedy na opisaniu każdego węzła grafu, przy czym kolejność opisywania jest dowolna.

Konstrukcja opisująca węzeł ma następującą strukturę:

[<*rezultat*> =] (<*operacja*> <*arg1*> <*arg2*> ... <*argn*>)

gdzie *rezultat* i *argumenty* to nazwy łuków grafu, zaś pole *operacja* zawiera nazwę operacji wykonywanej w węźle. Wynika z tego, że należy nadać łukom grafu wyróżniające je nazwy, aby móc opisać powiązania pomiędzy rezultatem jednej operacji i argumentem drugiej.

Programowa struktura symulatora jest otwarta, co pozwala na eksperymentowanie z dowolnie zdefiniowaną listą rozkazów (instrukcji), czemu musi, oczywiście, towarzyszyć opracowanie dla modułu wykonywania operacji biblioteki funkcji realizujących te rozkazy.

Wstępnie zdefiniowano dwa uniwersalne rozkazy o następujących nazwach operacji:

- *data* : nadaje zmiennej opisującej łuk wejściowy węzła wartość stałej podanej w polu argumentu; znacznik z wartością stałej jest tworzony w momencie inicjowania programu,
- *merge* : kieruje wartości znaczników danych z dwóch (lub więcej) łuków wejściowych (określonych argumentami) na wyjście węzła (określone jako "rezultat").

W pierwszym etapie symulator był testowany z listą rozkazów realizującą operacje algebraiczne.

Zasadnicza część badań dotyczyła natomiast zadań wyszukiwania danych. W tym celu zaprojektowano dwa rozkazy realizujące operacje algebry relacji: *selekcję* i *złączenie*. Ich składnia ma postać:

<*rezultat*> = select (*plik źródłowy, atrybuty, warunek selekcji, plik wynikowy*)

<*rezultat*> = join (*pierwszy plik źródłowy, drugi plik źródłowy, warunek łączenia, plik wynikowy*)

Wszystkie argumenty tych operacji są łańcuchami znaków, różna jednak jest ich zawartość i znaczenia. *Pliki źródłowe* to nazwy plików zawierających relacje będące argumentami operacji. *Plik wynikowy* to nazwa pliku, w którym zostanie zapisana relacja, będąca wynikiem operacji, przy czym *rezultat*, będący skróconą nazwą pliku, służy do oznaczania łuku wyjściowego węzła. *Warunek selekcji* i *warunek łączenia* mają znaczenie zgodne ze swoimi nazwami. Argument *atrybuty* zawiera listę atrybutów, które zostaną umieszczone w relacji wynikowej, co umożliwia jednoczesne wykonywanie operacji selekcji i projekcji. Możliwe jest również oddzielne wykonanie każdej z tych operacji, jeśli zajdzie taka potrzeba. Pusty *warunek selekcji* oznacza, że jest on zawsze prawdziwy i wykonywana jest wtedy tylko operacja projekcji. Argument *atrybuty* równy "*.all.*" powoduje, że przepisane do relacji wynikowej zostaną wszystkie atrybyty i realizowana operacja odpowiadać będzie selekcji.

Pomocniczym rozkazem dla zadań wyszukiwania danych jest (*erase*, <*p1*>, <*p2*>) służący do kasowania (roboczego) pliku *p1* pod warunkiem istnienia pliku *p2*.

5. Omówienie wykonanych eksperymentów

Symulator, wyposażony w bibliotekę realizującą funkcje odpowiadające opisanym rozkazom, umożliwił przeprowadzenie szeregu eksperymentów sprawdzających jego przydatność do rozpraszania obliczeń związanych z zadaniami wyszukiwania danych. Badania zostały przeprowadzone w lokalnej sieci 6 stacji roboczych Sun SPARCstation. Zbadano zależność zysku czasowego, wynikającego ze współbieżnej realizacji zadania, od ilości komputerów zaangażowanych w rozwiązywanie problemu.

Pierwsze z zrealizowanych zadań odpowiada pytaniu omówionemu poprzednio (rys. 1). Zapis tego zadania w opracowanym języku przedstawia się następująco:

```
egz=(data [s "egzam.dbf"])
prz=(data [s "przedm.dbf"])
styp=(data [s "stypen.dbf"])
stu=(data [s "studen.dbf"])
s1=(select egz [s "all."] [s "((termin='1') .and. (ocena >= '3.5'))"] [s "s1.dbf"])
s2=(select prz [s "nazwa='MATEMATYKA'"] [s "s2.dbf"])
s3=(select styp [s ".all."] [s "stypendium='S') .and. (miesiąc='10')"] [s "s.3.dbf"])
s4=(select stu [s ".all."] [s "(akademik<>' ') .and. (sredrok >= '3.3')"] [s4.dbf"])
j1=(join s1 s2 [s "1.przedmiot=2.przedmiot"] [s "j1.dbf"])
j2=(join s3 s4 [s "1.album=2.album"] [s "j2.dbf"])
wyn=(join j1 j2 [s "1.album=2.album"] [s "wyn.dbf"])
wynik=(select wyn [s "nazwisko, imię"] [s " "] [s "wynik.dbf"])
(erase wyn wynik)
(erase s1 j1)
(erase s2 j1)
(erase s3 j2)
(erase s4 j2)
(erase j1 wyn)
(erase j2 wyn)
end
```

Możliwy do osiągnięcia w tym zadaniu zakres operacji realizowalnych równolegle zilustrowano na rys. 2, w szczególności wszystkie wstępne operacje selekcji są niezależne, a więc potencjalnie współbieżne. Również pierwsze operacje złączenia mogą zostać wykonane niezależnie, po zakończeniu selekcji i utworzeniu odpowiednich relacji wejścio-

wych. Oczywiście, ostatnia operacja złączenia wykonywana jest na jednej stacji po wykonaniu wszystkich poprzedzających ją operacji, co pogarsza badany współczynnik przyspieszenia.

Tabela 1

Rezultaty realizacji zadania wyszukiwania danych na dwóch komputerach

Liczba komputerów	Zmierzony czas [s]	Współczynnik przyspieszenia
1	36.5	1.00
2	24.2	1.51

Przeprowadzone eksperymenty (tabela 1) wykazały, że maksymalny współczynnik przyspieszenia obliczeń (zdefiniowany jako iloraz czasów realizacji sekwencyjnej i równoległej danego programu), jaki udało się uzyskać w badanym przypadku, wyniósł 1.5 dla sytuacji, w której zadanie było rozwiązywane przy zaangażowaniu 2 stacji sieci.

Drugi z przeprowadzonych eksperymentów miał na celu zbadanie rezultatów, jakie można osiągnąć, w przypadku gdy realizowane są jednocześnie dwa zadania wyszukiwania dotyczące tej samej bazy danych. Do poprzednio rozwiązywanego zadania dołożono więc nowe: *podaj oceny zaliczeń z matematyki studentów, którzy znają język angielski na ocenę co najmniej 3.5 i nie pobierają stypendium naukowego*. Warto zwrócić uwagę na fakt, że w opisywanym eksperymencie oba pytanie korzystają z tych samych plików bazy danych, co może doprowadzić do sytuacji, w której dwa procesy będą pobierały informacje z tego samego zbioru dyskowego. System operacyjny Unix, w którego środowisku działa prezentowany program, umożliwia jednoczesny odczyt tego samego pliku przez kilka procesów, opisana sytuacja nie powinna mieć więc większego wpływu na osiągnięte rezultaty. Maksymalny współczynnik przyspieszenia zmierzony podczas realizacji takiego zadania wyniósł 1.73, w przypadku gdy zadania rozwiązywało 5 komputerów. Należy przy tym jednak zauważyć, że wzrost współczynnika przyspieszenia w miarę wzrostu liczby zaangażowanych stacji nie był zbyt znaczący.

Pełny zestaw otrzymanych czasów realizacji poprzedniego zadania, dla różnej liczby komputerów zaangażowanych w ich rozwiązywanie, został przedstawiony w tabeli 2.

Tabela 2

Rezultaty równoczesnej realizacji dwóch zadań wyszukiwania

Liczba komputerów	Czas realizacji [s]	Współczynnik przyspieszenia
1	170.5	1.00
2	111.8	1.53
3	106.8	1.60
4	104.9	1.63
5	98.5	1.73
6	98.8	1.73

6. Zakończenie

Przedstawiona w pracy koncepcja sterowania przepływem danych, będąca podstawą realizacji omówionego symulatora, umożliwia zapisanie w stosunkowo prostej formie zadań wyszukiwania danych realizowalnych równolegle. Podsumowując można stwierdzić, że osiągnięte rezultaty potwierdzają przydatność zaprezentowanego systemu do rozpraszania obliczeń w sieci komputerowej.

Literatura

[1] Barrett B.: AT&T s new signal processor will keep the fleet afloat into the next century. RECORD (AT&T Laboratories), Vol. 64, No. 2, 1986.

[2] Bieniok B.: Symulator komputera sterowanego przepływem danych. Praca dyplomowa. Instytut Informatyki. Politechnika Śląska, Gliwice 1993.

[3] Brown N. H.: The EMSP data flow computer. Proc. HICSS-17 Int. Conf. on System Sciences, 1984.

[4] Czajkowski G., Zieliński K.: Linda - środowisko do przetwarzania równoległego i rozproszonego w sieciach stacji roboczych, Informatyka nr 5, 1993.

[5] Dokumentacja systemu POSYBL--1.102.

[6] Gurd J. R.: The Manchester Dataflow Machine. Computer Physics Communications, 1985, No. 37.

[7] Hwang K., Briggs F. A.: Computer architecture and parallel processing. McGraw-Hill, New York 1984.

[8] Kozielski S., Szczerbiński Z.: Komputery równoległe: architektura, elementy programowania, WNT, Warszawa 1993.

[9] Kozielski S., Piec J., Grzywocz J.: System wyszukiwania danych oparty na modelu relacji uniwersalnej, Archiwum Informatyki Teoretycznej i Stosowanej, 1993 (w druku).

[10] Ullman J.D.: Principles of Database Systems. Computer Science Press, Rockville, 1983. Polskie wydanie: Systemy baz danych. WNT, Warszawa 1988.

[11] Ullman J.D.: Database and Knowledge-Base Systems. Computer Science Press, Rockville, 1989.

[12] Wilk A.: Wykorzystanie sieci NetWare firmy Novell do tworzenia oprogramowania współbieżnego na przykładzie implementacji równoległego algorytmu rozwiązania uproszczonego problemu Hilberta, Gliwice 1990 (opracowanie niepublikowane).

Recenzent: Dr inż. Andrzej Wilk

Wpłynęło do redakcji 27.09.1993 r.

Abstract

First part of the paper is devoted to a discussion of such a manner of data retrieving tasks formulation, which enables computation distribution in a computer network. The relational algebra has been chosen as a proper level of query formulation. Two variants of distribution control have been considered. Advantages of parallel computing according to the idea of dataflow control have been appointed.

An elaborated simulator of dataflow computer is presented in the second part of the paper. The simulator consists of two processes: a manager (data tokens matching, instructions completing) and a executor (intructions execution). Relational algebra operations are performed as simulator instructions in the executor. The executor may be started on several network stations simultaneously, that enables parallel instructions realization.

Results of data retrieving experiments are presented in the last part of the paper.