ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1994

Seria: INFORMATYKA z. 26 Nr kol. 1255

Piotr BAJERSKI

REALIZACJA ALGORYTMÓW W SYSTEMIE STEROWANYM PRZEPŁYWEM ARGUMENTÓW W SIECI LOKALNEJ*

Streszczenie. W artykule przedstawiono realizację systemu sterowanego przepływem argumentów w sieci stacji roboczych Sun. Zaimplementowany system wykorzystano do równoległego wykonania kompilacji.

ALGORITHMS IMPLEMENTATION IN THE ARGUMENT FLOW DRIVEN SYSTEM IN A LOCAL NETWORK

Summary. The article presents implementation of the Argument Flow Driven System in a Sun workstation network. The system is used for a parallel compilation.

DIE REALISIERUNG DER ALGORITHMEN IN DEM DURCH DEN ARGUMENTENFLUß GESTEUERTEN SYSTEM IM LOKALEN NETZWERK

Zusammenfassung. Im Artikel wurde die Realisierung des durch den Argumentenfluß gesteuerten Systems im Netzwerk der SUN-Rechner vorgestellt. Das implementierte System wurde zu der parallelen Ausführung der Compilierung ausgenutzt.

* Praca zrealizowana w ramach Projektu Badawczego (GRANTU) nr KBN 3 P406 011 04.

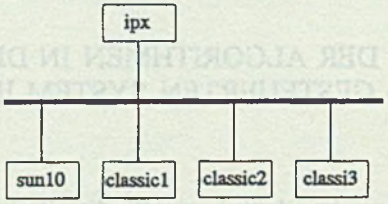
1. Wprowadzenie

W pracy [1] podano ogólną koncepcję systemu komputerowego sterowanego przepływem argumentów. Celem niniejszej pracy była analiza i rozwinięcie tej koncepcji na podstawie badań doświadczalnych. Otrzymano potwierdzenie możliwości uzyskania przyśpieszenia dla przypadków zakładanych w [1]. Wskazano również i potwierdzono doświadczalnie możliwość realizacji systemów sterowanych przepływem argumentów przy ograniczonej liczbie procesorów operacyjnych.

W celu uściślenia terminologii w dalszej części artykułu termin kompilacja określa proces translacji plików zawierających kod źródłowy do postaci plików typu object. Termin linkowanie jest używany do nazwania procesu łączenia plików typu object wraz z bibliotekami w jeden plik wykonywalny. Potrzebne do utworzenia pliku wykonywalnego kompilacje i linkowanie są określane terminem translacja.

2. Konfiguracja sprzętowa systemu

Eksperymenty prowadzone były na komputerach Sun połączonych siecią Ethernet. Dla dalszych rozważań ważne jest, że sieć ta jest siecią lokalną, w której transmisja odbywa się metodą rozgłoszeniową, a czas transmisji ramki przez sieć jest zmienną losową (zależy od obciążenia sieci).



Rys. 1. Konfiguracja sieci lokalnej
Fig. 1. The local network configuration

Spośród siedmiu dostępnych Sun'ów wykorzystano pięć mających zbliżoną moc obliczeniową (rys. 1). Cztery komputery wykonywały procesy realizujące operacje syste-

mu sterowanego przepływem argumentów (SSPA), a piąty zarządzał wykonaniem całego programu.

Komputery pracują pod kontrolą sunowskiej wersji UNIX-a i korzystają z sieciowego systemu plików NFS. Systemy plików są tak zamontowane na każdym z komputerów, że użytkownik widzi to samo drzewo kartotek na każdym z nich. Kartoteki lokalne i zdalne są widziane tak samo przez programy, a system NFS zajmuje się fizyczną transmisją plików między komputerami, jeżeli wystąpi odwołanie do pliku zdalnego.

3. Wykorzystane narzędzia programowe

Model Lindy jest modelem dzielonej pamięci asocjacyjnej, nazywanej przestrzenią krotek. Linda udostępnia operacje umożliwiające wstawianie i pobieranie danych z tej przestrzeni. Dane te, nazywane krotkami, składają się z ciągów pól. Wyróżnia się dwa rodzaje krotek: bierne i aktywne. Bierne są ciągami wartości. Aktywne krotki zawierają wywołania funkcji, które są wartościowane jako osobne procesy równoległe względem siebie nawzajem i innych procesów wykonujących się w środowisku Lindy. Aktywne krotki wymieniają dane przez przestrzeń krotek, generując, odczytując i konsumując bierne krotki. Aktywna krotka po zakończeniu się procesów, które zostały utworzone przy jej wartościowaniu, staje się krotką bierną.

Do zrealizowania obliczeń równoległych potrzebne są dwa języki: język pozwalający zapisać obliczany problem i język umożliwiający tworzenie nowych procesów i ich koordynację. W niniejszej pracy pierwszym językiem jest język C, drugim język C-Linda, będący implementacją modelu Linda.

Język C-Linda jest zanurzony w języku C i dodaje do niego operacje wstawiania i pobierania krotek do/z przestrzeni krotek.

Wstawianie krotki biernej jest dokonywane operacją **out**. Do wstawienia aktywnej krotki służy operacja **eval**.

Pobieranie krotki jest dokonywane operacjami **in** i **rd**. Krotka jest pobierana na zasadzie dopasowania do wzorca, będącego argumentem tych operacji. Operacja **in** pobiera krotkę z przestrzeni krotek z równoczesnym usunięciem jej, operacja **rd** odczytuje krotkę (krotka pozostaje w przestrzeni krotek po wykonaniu operacji). Jeżeli znaleziono krotkę pasującą do wzorca, to za argumenty formalne podstawiane są argumenty aktualne znalezionej krotki. Jeżeli nie znaleziono pasującej krotki, wykonanie procesu jest wstrzymywane do chwili pojawienia się w przestrzeni krotek pasującej krotki. Istnieją

wersje predykatowe operacji **in** i **rd** nie blokujące wykonania programu, nazywające się odpowiednio: **inp** i **rdp**.

Przykłady operacji języka C-Linda:

int i;

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.

4. Realizowany algorytm

Zaimplementowano algorytm translacji programu napisanego w języku C na równoważny mu program wykonywalny. Przy zrównoleglaniu problemu wykorzystano fakt, że język C zapewnia rozłączną kompilację, tzn. każdy z modułów programu jest kompilowany oddzielnie, a następnie wyniki kompilacji są łączone w jeden program wykonywalny. Widać tutaj naturalne źródło równoległości. Każdy z modułów źródłowych może się kompilować w tym samym czasie na innej maszynie, a dopiero przy linkowaniu wszystkie pliki typu object są przesyłane na jeden komputer. Duży wpływ na otrzymane przyśpieszenie ma fakt, że kompilacja trwa znacznie dłużej niż linkowanie.

Przyjęto, że szyna SSPA jest reprezentowana przez krotkę, a operacja przez aktywną krotkę zawierającą wywołanie funkcji realizującej tę operację. Przepływ danych i zmiennych jest zapisany na sztywno w kodzie programu.

$$\begin{aligned} M_1^* &= C(M_1) \\ M_2^* &= C(M_2) \\ M_3^* &= C(M_3) \\ P^* &= L(M_1^*, M_2^*, M_3^*) \end{aligned} \tag{1}$$

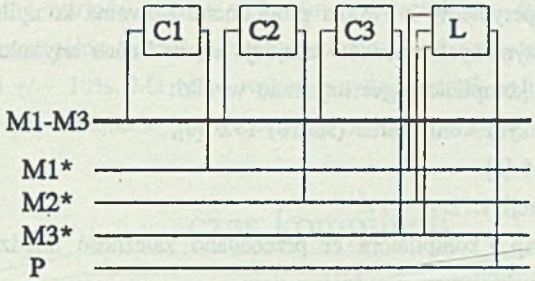
Na rys. 2 przedstawiono system sterowany przepływem argumentów dla problemu translacji. Przez C oznaczono proces kompilacji przez L proces linkowania. Przez M_i oznaczono moduł źródłowy o numerze i. Przez M_i^* skompilowany moduł M_i. Program wynikowy (wykonywalny) jest oznaczony przez P. Równanie 1 przedstawia formę kanoniczną odpowiadającą systemowi z rys. 2.

Na rys. 3 przedstawiono macierz zmiennych formy kanonicznej. Jak widać, jest to macierz górna bez głównej przekątnej, co zapewnia poprawność działania systemu.

Danymi dla SSPA są ścieżki dostępu do plików. Są one wyprowadzane do przestrzeni krotek przez program zarządzający.

Algorytm procesu wykonującego kompilację:

1. Odczytaj ścieżkę dostępu do pliku źródłowego z przestrzeni krotek.
2. Skompiluj plik źródłowy, gdy zostaną zgromadzone wszystkie argumenty.
3. Wyprowadź ścieżkę dostępu do utworzonego pliku typu object.



Rys. 2. System sterowany przepływem argumentów dla problemu translacji
Fig. 2. The Argument Flow Driven System for the translation problem

Algorytm procesu wykonującego linkowanie:

1. Pobieraj w pętli z przestrzeni krotek do tablicy ścieżki dostępu do plików typu object.
2. Wykonaj linkowanie modułów, gdy zostaną zgromadzone wszystkie argumenty.
3. Wyprowadź komunikat o zakończeniu procesu translacji.

Procesy kompilacji i proces linkowania są uruchamiane przez program główny.

	M1*	M2*	M3*	P
M1*				X
M2*				X
M3*				X
P				

Rys. 3. Macierz zmiennych formy kanonicznej
Fig. 3. The variable matrix of canonical form

Prowadzono eksperymenty na programach użytkowych ściągniętych przez sieć Internet. Gdy programy miały większą liczbę modułów niż wynosiła liczba stanowisk reali-

zujących operację kompilacji w zaimplementowanym SSPA, szybkość translacji badano na największych modułach, a resztę dołączano podczas linkowania.

5. Otrzymane wyniki

Przeprowadzono eksperymenty z dwoma programami i dwoma kompilatorami. Tabelaryczne zestawienie otrzymanych wyników znajduje się na końcu artykułu (tabela 2).

Dla programu **gzip** i kompilatora **gcc** uzyskano wyniki:

— translacja na najszybszym komputerze (sun10) 19.6 [s],

— translacja w sieci 11.4 [s],

— przyśpieszenie translacji 1.72.

Dla programu **gifstrip** i kompilatora **cc** przebadano zależność między przydziałem zadań na procesory a uzyskiwanym przyśpieszeniem.

Poniżej przedstawiono czasy trwania poszczególnych operacji SSPA dla przykładowego wykonania programu:

Tabela 1

Przykładowe czasy kompilacji i linkowania

Komputer	Wykonywane zadanie	Czas wykonania [s]
sun10	kompilacja modułu 1	7.28
classic1	kompilacja modułu 3	4.55
classic2	linkowanie	1.01
classic3	kompilacja modułu 2	5.68
ipx	komputer sterujący	

Z tabeli 1 wynika, że najwięcej czasu procesora zajmuje kompilacja modułu numer 1. Kompilując ten moduł na najszybszym komputerze (sun10) uzyskano wyniki:

— translacja na najszybszym komputerze (sun10) 15.2 [s],

— translacja w sieci 8.33 [s],

— przyśpieszenie translacji 1.82.

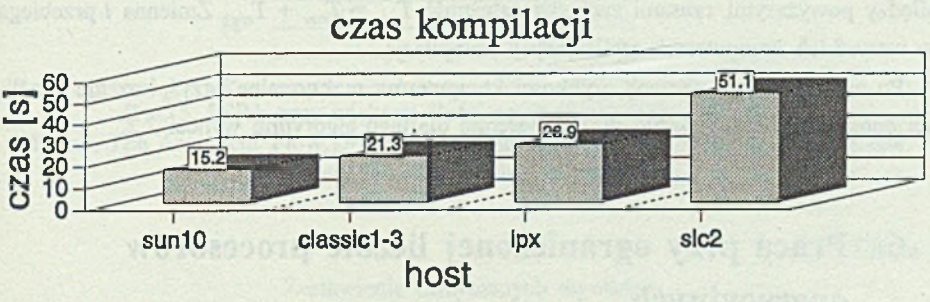
Jeżeli moduł1 był kompilowany na którymś z komputerów typu classic, czas translacji całości wzrastał do 11.1 [s], a przyśpieszenie translacji spadało do 1.37.

Jeżeli dokonano translacji wyłączając sun10 z obliczeń, przyśpieszenie wzrosło do 23.4 [s] / 12.9 [s] = 1.81. Wyłączenie komputera sun10 wyrównywało różnice szybkości

pomiędzy używanymi komputerami, a najszybszymi komputerami stawały się komputery classic1-3.

Na wartość otrzymanego przyśpieszenia duży wpływ ma nierówny rozmiar podzadań, wykonywanych równolegle. Moduł 1 stanowi połowę całego zadania kompilacji, co powoduje, że pozostałe procesory realizujące kompilacje czekają bezczynnie po zakończeniu swoich zadań.

Przedstawione wartości czasów wykonania są wartościami średnimi z wielu wykonań programu. Poszczególne wartości czasów odbiegają od wartości średnich, nie przekraczając na ogół +/− 10%. Ma na to wpływ przydział zadań na poszczególne komputery, ruch w sieci i inne zadania wykonujące się w tym samym czasie.



Rys. 4. Szybkość translacji na wykorzystywanych komputerach
Fig. 4. Speed of translation on the computers used

Aby wypowiedzieć się na temat otrzymanego przyśpieszenia należy rozważyć różne szybkości użytych komputerów. Poniżej przedstawiono czasy translacji programu **gifstrip** na poszczególnych komputerach i w nawiasach stosunek tych czasów do czasu translacji na najszybszym komputerze (sun10).

sun10 15.2 [s] (1)
classic1-3 21.3 [s] (0.71)
ipx 26.9 [s] (0.57)
slc2 51.1 [s] (0.3).

Wartości te zostały zilustrowane na rys. 4.

Aby oszacować teoretyczne przyśpieszenie zmodyfikowano wzór Amdahla tak, aby uwzględniał różną szybkość wykorzystywanych komputerów (wzór 2).

$$S < \frac{T_{\min}}{T_{szer} + \frac{T_{rów}}{\sum_i \frac{T_{\min}}{T_i}}}, \qquad (2)$$

gdzie:

T_{min} — czas wykonania programu na najszybszym komputerze,

T_{szer} — czas wykonania tej części programu, której nie da się zrównoleglić (w tym przypadku jest to linkowanie),

$T_{rów}$ — czas wykonania tej części programu, która da się zrównoleglić (w tym przypadku jest to kompilacja).

Między powyższymi czasami zachodzi zależność $T_{min} = T_{szer} + T_{rów}$. Zmienna i przebiega po wszystkich komputerach realizujących kompilację.

Po uwzględnieniu różnych szybkości komputerów maksymalne przyśpieszenie możliwe teoretycznie do uzyskania w tym systemie dla tego algorytmu wynosi 2.2.

6. Praca przy ograniczonej liczbie procesorów operacyjnych

Przy translacji wszystkie operacje w SSPA poza ostatnią wykonują taką samą funkcję — kompilację. Mając ograniczoną liczbę procesorów można zaimplementować SSPA tak, że kilka operacji takiego samego typu jest realizowanych przez ten sam proces. Na rys. 5 przedstawiono SSPA dla takiego przypadku. Dane znajdują się w kolejce, a procesy C1, C2, C3 tworzą wielokanałowe stanowisko obsługi.

Przeprowadzono eksperyment z programem **chils**, składającym się z 19 modułów, liczących od kilku tysięcy do kilkudziesięciu linii kodu. W tym przypadku translacji podlegały wszystkie moduły. Każdy z procesów dokonujących kompilacji w pętli pobierał nazwę programu, kompilował go i wysyłał wyniki. Pętla kończyła się, gdy nie było już zadań do skompilowania. Dzięki dużej ilości zadań przypadającej na każdy z procesów realizujących operacje SSPA uzyskano dynamiczne równoważenie obciążenia.

Dla konfiguracji, w której ipx sterował wykonaniem procesów, otrzymano rezultaty:

— translacja na najszybszym komputerze (sun10) 43.4 [s],

— translacja w sieci 21.2 [s],

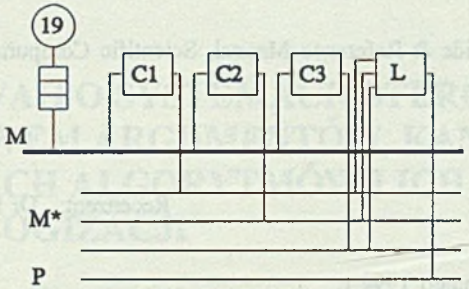
— przyśpieszenie translacji 2.05.

Jeżeli nie uwzględniać komputera sun10 (bardziej zbliżone moce obliczeniowe wykorzystywanych komputerów), wyniki przedstawiają się następująco:

— translacja na najszybszym komputerze (classic) 64.98 [s],

— translacja w sieci 26.9 [s],

— przyśpieszenie translacji 2.42.



Rys. 5. SSPA przy większej ilości operacji niż liczba procesorów

Fig. 5. The Argument Flow Driven System in case when the number of operations is greater than the number of processors

Tabela 2

Zestawienie otrzymanych wyników

Lp.	Program	Czas kompilacji [s]		Przyśpieszenie	Uwagi
		min	w sieci		
1	gzip	19.6	11.4	1.72	
2	gifstrip	15.2	8.3	1.82	Moduł1 na komputerze sun10
3	gifstrip	15.2	11.1	1.37	Moduł1 na komputerze classic
4	gifstrip	23.4	12.9	1.81	Bez komputera sun10
5	chils	43.4	21.2	2.05	19 modułów z sun10
6	chils	65.0	26.9	2.42	19 modułów bez sun10

LITERATURA

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

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

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

Recenzent: Dr inż. Andrzej Wilk

Wpłynęło do Redakcji 2 lutego 1994 r.

Abstract

In [1] are shown the theoretical bases of the Argument Flow Driven System. This article deals with implementation of such a system in a local network and using it for parallel compilation. It is shown, in section 4, that the variable matrix of canonical form of the compilation algorithm is the upper triangular one with the main diagonal excluded. This is the sufficient condition for realization of the Argument Flow Driven System.

Section 5 shows experimental results, proving that the process of compilation can be speeded up by using the Argument Flow Driven System. The results obtained are shown in table 2.

In the last section it is shown how to realize the Argument Flow Driven System for the case when there are more operations than available processors.