ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: INFORMATYKA z. 34 Nr kol. 1381

Mariusz FRAŚ
Politechnika Wrocławska, Instytut Sterowania i Techniki Systemów

RÓWNOLEGŁA REALIZACJA REKURENCYJNYCH ALGORYTMÓW LOGICZNO-ALGEBRAICZNYCH

Streszczenie. W artykule przedstawiono algorytm równoległej realizacji obliczeń metodą logiczo-algebraiczną dla zadań z reprezentacją wiedzy w formie faktów. Omówiono istotę rekurencyjnych procedur obliczeń z dekompozycją faktów. Zaproponowano sposób obliczeń w systemie wieloprocesorowym i zaprezentowano algorytm równoległy równoważący obciążenie procesorów. Przedstawiono wyniki eksperymentów obliczeniowych dla wybranych zadań.

PARALLEL EXECUTION OF RECURSIVE LOGIC-ALGEBRAIC ALGORITHMS

Summary. The paper presents algorithm of parallel computations for the logic-algebraic method applied to a class of knowledge-based systems. The idea of recursive procedure based on the decomposition of facts is presented. The parallel computations approach is suggested and dynamic load balancing algorithm is proposed. Some results of computational experiments are also discussed.

1. Wstęp

Wiele z zadań obliczeniowych w sposób naturalny nadaje się do realizacji w środowisku równoległego przetwarzania. W dobie szybkiego rozwoju lokalnych sieci komputerowych oraz coraz większej dostępności komputerów o architekturze wieloprocesorowej realizacja algorytmów obliczeniowych w sposób równoległy staje się coraz powszechniejsza. Efektywność algorytmów równoległych zależy między innymi od równomiernego obciążenia procesorów zadaniami obliczeniowymi. W wielu przypadkach niezbędne jest stosowanie dodatkowej

procedury mającej na celu równoważenie obciążenia dla obliczeń w systemie wieloprocesorowym.

W niniejszym artykule przedstawiono wyniki dotychczasowych badań mających na celu adaptację logiczno-algebraicznych algorytmów rozwiązywania zadań, zastosowanych w istniejących systemach ekspertowych CONTROL-LOG/S i CLASS-LOG/S, do wieloprocesorowych systemów komputerowych.

1.1. Wprowadzenie do metody logiczno-algebraicznej

Systemy z reprezentacją wiedzy mogą być opisywane w różny sposób. Reprezentację wiedzy RW w formie faktów można w skrócie określić [1, 3] jako zestaw formuł elementarnych α reprezentujących pewne własności opisywanego obiektu oraz faktów - formuł logicznych opisujących zależności pomiędzy formułami elementarnymi za pomocą podstawowych operatorów logicznych:

$$RW = \langle \alpha, F(\alpha) \rangle \tag{1}$$

gdzie: α - ciąg formuł elementarnych,

F - ciąg faktów (iloczyn logiczny).

Zakłada się, że wszystkie fakty podane przez eksperta są prawdziwe. Wśród formuł elementarnych możemy wyróżnić formuły elementarne wejściowe α_x i formuły elementarne wyjściowe α_y. Analogicznie wyróżniamy fakty wejściowe $F_x(\alpha_x)$ i wyjściowe $F_y(\alpha_y)$.

Dla tak sformułowanej reprezentacji wiedzy rozpatruje się dwa podstawowe zadania: zadanie analizy i zadanie podejmowania decyzji. Zadanie analizy polega na odpowiedzi na pytanie o wartość zdania F_y (czy jest prawdziwe, fałszywe lub nie potrafimy o nim nic powiedzieć). Zadanie podejmowania decyzji polega na wyznaczeniu najlepszego F_x, dla którego spełniona jest implikacja $F_x \Rightarrow F_y$.

Do rozwiązywania zadań analizy i podejmowania decyzji stosuje się opartą na gruncie algebry logiki **metodę logiczno-algebraiczną** [2, 3]. Szczególną realizacją tej metody są **rekurencyjne algorytmy z dekompozycją zbioru faktów** szczegółowo opisane w [2]. Dekompozycja polega na specjalnym podziale (dekompozycji) zbioru faktów bazy wiedzy na odpowiednie podzbiory:

$$F(a) = \overline{F}_1(\overline{a}_0, \overline{a}_1) \wedge \ldots \wedge \overline{F}_j(\overline{a}_{j-1}, \overline{a}_j) \wedge \ldots \wedge \overline{F}_J(\overline{a}_{J-1}, \overline{a}_J) \tag{2}$$

gdzie:

$\overline{F}_j$ - odpowiedni podciąg (iloczyn logiczny) faktów dla stopnia j, $j=1,\ldots,J$,

$\bar{a}_{j-1}$, $\bar{a}_j$ - odpowiednie podciągi wartości logicznych formuł elementarnych,

J - liczba ciągów faktów- stopni dekompozycji.

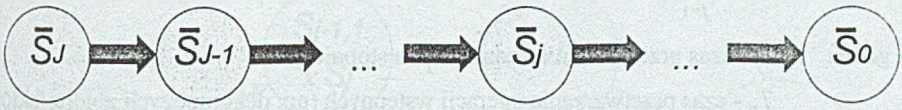
Procedura rekurencyjna polega na wyliczaniu na kolejnych stopniach odpowiednich podzbiorów $\bar{S}_j$ zbiorów S_j wszystkich ciągów $\bar{a}_j$ według formuły:

$$\bar{S}_{j-1} = \left\{ \bar{a}_{j-1} \in S_{j-1} : \bigvee_{\bar{a}_j \in \bar{S}_j} \left[\bar{F}_j\left(\bar{a}_{j-1}, \bar{a}_j\right) = 1 \right] \right\} \tag{3}$$

gdzie:

$j = J, J\text{-}1, \ldots, 1$, $\bar{S}_J = S_J$.

W efekcie cykl obliczeń można przedstawić jako wyznaczanie kolejno zbiorów $\bar{S}_j$:



Rys. 1. Cykl obliczeniowy algorytmu rekurencyjnego
Fig. 1. The computation scheme of recursive algorithm

Zawartość zbioru wyliczonego na ostatnim stopniu ($\bar{S}_0$ dla metody w przód i $\bar{S}_J$ dla metody wstecz) jest podstawą do rozwiązania zadania (udzielenia odpowiedzi lub podjęcia decyzji przez system ekspertowy).

W celu wyznaczenia zbioru $\bar{S}_{j-1}$ trzeba wykonać $\gamma_j = \beta_j \cdot \delta_j$ elementarnych kroków obliczeniowych polegających na sprawdzeniu wartości faktu $\bar{F}_j(\bar{a}_{j-1}, \bar{a}_j)$, gdzie:

$\beta_j = 2^{b_j}$ - liczność zbioru S_{j-1} , b_j - długość ciągu $\bar{a}_{j-1}$,

δ_j - liczność zbioru $\bar{S}_j$ (liczba ciągów, które przeszły do obliczeń z poprzedniego stopnia).

Po pewnych usprawnieniach [5] liczbę elementarnych kroków obliczeniowych wykonywanych na stopniu j można zmniejszyć do:

$$\gamma'_j = \beta'_j \cdot \delta'_j \le \gamma_j \tag{4}$$

gdzie:

β'_j - liczba sprawdzanych kombinacji wartości ciągu $\bar{a}_{j-1}$ ze zbioru S_{j-1},

δ'_j - średnia z liczby sprawdzanych ciągów $\bar{a}_j$ ze zbioru $\bar{S}_j$ dla poszczególnych $\bar{a}_{j-1}$.

Konsekwencją wprowadzonej dekompozycji jest podział całego zadania obliczeniowego Z, którym jest rozwiązywanie zadania podstawowego, na J zadań obliczeniowych $Z_j, j \in \overline{1,J}$ wykonywanych sekwencyjnie:

Rys. 2. Schemat wykonywania zadań obliczeniowych w procedurze rekurencyjnej na pojedynczym procesorze

Fig. 2. The scheme of task execution on single processor for recursive procedure

Całkowity czas obliczeń dla zadania Z w przypadku wykonywania na pojedynczym procesorze, kiedy obliczenia na poszczególnych stopniach nie nakładają się na siebie, wynosi:

$$T_c = T_p + \sum_{j=1}^{J} T_j + T_k \tag{5}$$

gdzie: T_j - czas przetwarzania zadania Z_j na stopniu j,

T_p - czas przetwarzania operacji wstępnych (np. dekompozycji zbioru faktów),

T_k - czas przetwarzania operacji końcowych - wyznaczenie rozwiązania na podstawie zawartości zbioru $\overline{S}_0$.

W przypadku dużych zadań Z czasy T_p i T_k możemy uznać za pomijalnie małe. Czas obliczeń T_j na danym stopniu jest wprost proporcjonalny do liczby elementarnych kroków obliczeniowych γ'_j.

2. Równoległa realizacja algorytmu rekurencyjnego

Pomimo pewnych usprawnień rekurencyjnego algorytmu logiczno-algebraicznego, dla dużych wartości γ_j czas obliczeń zadania może być znacząco duży. Dysponując wieloprocesorowym środowiskiem przetwarzania danych można zmniejszyć ten czas poprzez równoległe wykonywanie obliczeń.

Proponuje się, aby na każdym stopniu j zbiór $S_{j-1} \times \overline{S}_j$ sprawdzanych ciągów $(\overline{a}_{j-1}, \overline{a}_j)$ rozbić wyczerpująco na P_j rozłącznych podzbiorów $(S_{j-1} \times \overline{S}_j)_p$, $p \in \overline{1,P_j}$ i dla tych podzbiorów wykonywać obliczenia w sposób równoległy. Ze względu na sposób uporządkowania przeglądania ciągów zbioru $S_{j-1} \times \overline{S}_j$ [5] najprostszy i najpraktyczniejszy jest podział

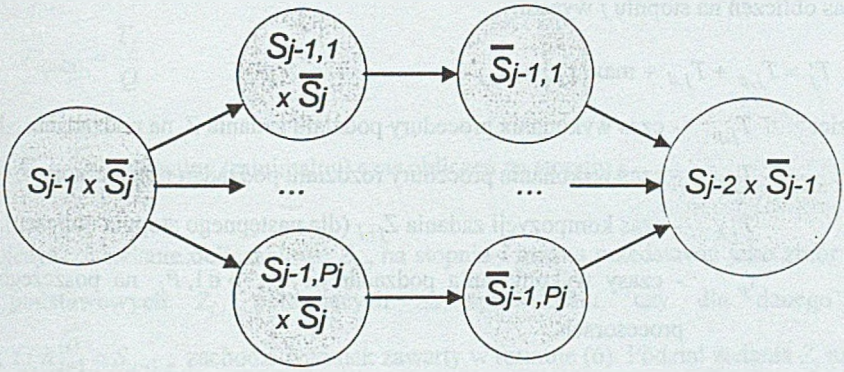
tego zbioru poprzez podział zbioru S_{j-1} na P_j rozłącznych zbiorów $S_{j-1,p}$, $p \in \overline{1,P_j}$. Zatem na danym stopniu j otrzymuje się P_j rozłącznych zbiorów $S_{j-1,p} \times \overline{S}_j$, $p \in \overline{1,P_j}$, dla których dokonuje się obliczeń analogicznie do formuły (3):

$$\overline{S}_{j-1,p} = \left\{ \overline{a}_{j-1} \in S_{j-1,p} : \bigvee_{\overline{a}_j \in \overline{S}_j} \left[\overline{F}_j\left(\overline{a}_{j-1}, \overline{a}_j\right) = 1 \right] \right\} \tag{6}$$

Zbiór $\overline{S}_{j-1}$ jest sumą zbiorów $\overline{S}_{j-1,p}$, $p \in \overline{1,P_j}$.

$$\overline{S}_{j-1} = \bigcap_{p \in \overline{1,P_j}} \overline{S}_{j-1,p} \tag{7}$$

Schemat wyznaczania kolejnych zbiorów na stopniu j przedstawiono na rys. 3.

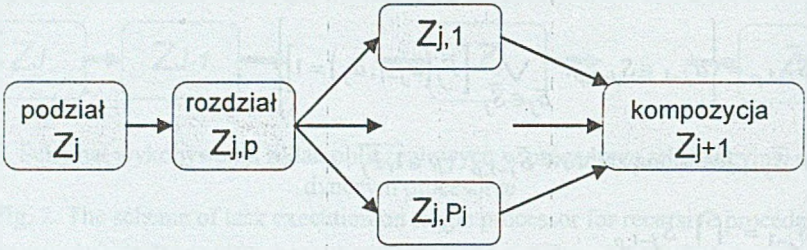


Rys. 3. Schemat generacji zbiorów na stopniu j dla algorytmu równoległego
Fig. 3. The scheme of generation of sets on step j for parallel algorithm

Zauważmy, że zbiory $\overline{S}_{j-1,p}$, $p \in \overline{1,P_j}$ są rozłączne, co znacznie upraszcza i przyspiesza proces generowania zbioru $S_{j-2} \times \overline{S}_{j-1}$ dla następnego stopnia obliczeń. Proponuje się, aby podział spełniał warunek równej liczności zbiorów $S_{j-1,p}$ oraz by liczba zbiorów nie przekraczała liczby Q dostępnych procesorów w systemie. Procedura obliczeniowa na stopniu j przedstawia się następująco:

1. Podział zadania Z_j na P_j podzadań $Z_{j,p}$, $p \in \overline{1,P_j}$.
2. Rozdział podzadań $Z_{j,p}$ do wykonania na Q procesorów.
3. Wykonanie równoległe obliczeń na poszczególnych procesorach wg formuły (6).

4. Kompozycja zadania Z_{j+1} - zebranie wyników obliczeń podzadań $Z_{j,p}$ (zbiorów $\bar{S}_{j-1,p}$) i sformułowanie zadania obliczeniowego Z_{j+1}.



Rys. 4 Schemat wykonywania zadań na stopniu j w systemie wieloprocesorowym
Fig. 4. The scheme of task execution on step j in multiprocessor enviroment

Czas obliczeń na stopniu j wynosi:

$$T_j' = T_{j,p} + T_{j,d} + \max\{\tau_p\} + T_{j,k} \tag{8}$$

gdzie: $T_{j,p}$ - czas wykonania procedury podziału zadania Z_j na podzadania,

$T_{j,d}$ - czas wykonania procedury rozdziału podzadań na procesory,

$T_{j,k}$ - czas kompozycji zadania Z_{j+1} (dla następnego stopnia obliczeń),

τ_p - czasy wykonywania podzadań $Z_{j,p}$, $p \in \overline{1, P_j}$ na poszczególnych procesorach.

Jeśli założymy proste metody podziału, nie wprowadzające dodatkowego narzutu czasowego, oraz środowisko obliczeniowe, w którym czas komunikacji międzyprocesorowej jest mały w porównaniu z czasem obliczeń, wtedy czas obliczeń można oszacować jako:

$$T_j' = \max_p \{\tau_p\}, \quad p \in \overline{1, P_j} \tag{9}$$

3. Algorytm równoważenia obciążenia

3.1. Obciążenie procesorów obliczających podzadania

Rozpatrzmy następujący przypadek, w którym:

1. Dysponujemy środowiskiem Q procesorów identycznych.

2. $Q=P_j$ - wykonujemy podział zadania Z_j na tyle podzadań $Z_{j,p}$, $p \in \overline{1,P_j}$, ile jest dostępnych procesorów.
3. Procesory wykonują tylko przydzielone przez nas zadania.
4. Czas komunikacji międzyprocesorowej jest pomijalnie mały.

Całkowity czas wykonania równolegle obliczanego zadania Z_j na stopniu j wynosi:

$$T'_j = \max_p \left\{\tau_p\right\}, \quad p \in \overline{1,Q} \tag{10}$$

Natomiast czas wykonania na pojedynczym procesorze wynosi:

$$T_j = \sum_{p=1}^{Q} \tau_p \tag{11}$$

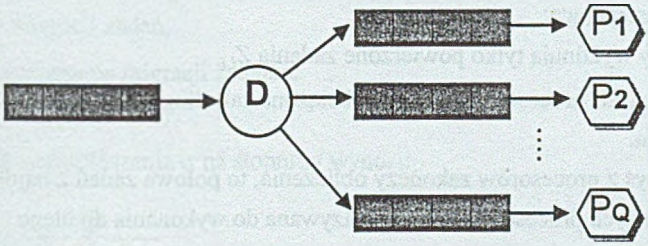
Gdy $\tau_1 = \tau_2 = ... = \tau_Q$, wtedy otrzymujemy Q-krotne skrócenie czasu obliczeń na stopniu j:

$$T'_{j(\min)} = \frac{T_j}{Q} \tag{12}$$

gdzie:

$T'_{j(\min)}$ - optymalny (minimalny) czas obliczeń na stopniu j.

Pojedyncze zadanie obliczeniowe $Z_{j,p}$ na stopniu j można przedstawić jako zbiór β'_j zadań **podstawowych** $Z_{j,i}$ polegających na sprawdzeniu, czy dla danego ciągu $\bar{a}^{(i)}_{j-1}$, $i : \bar{a}^{(i)}_{j-1} \in S_{j-1,p}$ zachodzi warunek zawarty w formule (6). Podział zadania Z_j na $P_j=Q$ podzadań $Z_{j,p}$ można traktować jako rozdział β'_j zadań podstawowych $Z_{j,i}$ do wykonania równolegle na Q procesorach (rys. 5).



Rys. 5. Model równoległego wykonywania zadań jako zbioru zadań podstawowych
Fig. 5. Model of parallel task execution as a set of basic tasks

Czas wykonania zadania $Z_{j,p}$, $p \in \overline{1,Q}$ jest sumą czasów wykonania zadań $Z_{j,i}$, $i : \bar{a}_{j-1}^{(i)} \in S_{j-1,p}$:

$$\tau_p = \sum_i t_{j,i} \ , \quad i : \bar{a}_{j-1}^{(i)} \in S_{j-1,p} \tag{13}$$

gdzie:

$t_{j,i}$ - czas wykonania zadania $Z_{j,i}$ na jednym procesorze.

Czasy $t_{j,i}$ wykonywania poszczególnych zadań podstawowych $Z_{j,i}$ mogą się znacznie różnić między sobą [5]. Postulowany uprzednio warunek równej liczności zbiorów $S_{j-1,p}$, czyli równej długości kolejek $q \in \overline{1,Q}$ zadań wykonywanych na poszczególnych procesorach, nie gwarantuje równych czasów wykonywania zadań $Z_{j,p}$. Zatem czas wykonania obliczeń zadania Z_j na stopniu j (i w konsekwencji całego zadania obliczeniowego Z) może być większy od optymalnego czasu realizacji:

$$T_j' \geq \frac{T_j}{Q} \tag{14}$$

Minimalizację czasu obliczeń można sprowadzić do zadania równoważenia obciążenia Q procesorów w systemie wieloprocesorowym:

$$\min \max_p \tau_p \ , \ p = 1, 2, \ldots, Q \tag{15}$$

3.2. Algorytm

Zaproponowano i zrealizowano algorytm dynamicznego równoważenia obciążenia oparty na migracji zadań z procesorów obciążonych do procesorów nieobciążonych. Założenia do algorytmu są następujące:

1. Procesory wykonują tylko powierzone zadania $Z_{j,i}$.
2. Uznajemy, że procesory są równo obciążone tak długo, dopóki wszystkie wykonują obliczenia.
3. Jeśli któryś z procesorów zakończy obliczenia, to połowa zadań z najdłuższej kolejki z kolejek innych procesorów jest przekazywana do wykonania do niego.

Poniżej przedstawiono szczegółowo równoległy algorytm obliczeń dynamicznie równoważący obciążenie w systemie. Kroki 4 do 13 (pomiędzy liniami ciągłymi) wykonują się równolegle na Q procesorach.

Algorytm równoważenia obciążenia:

1. $j = J$.
2. Podziel zadanie Z_j na zadania podstawowe $Z_{j,i}$.
3. Przekaż zadania podstawowe po równo do wykonania na Q procesorów.

---

4. Pobierz zadanie $Z_{j,i}$ ze swojej kolejki do wykonania.
5. Wykonaj zadanie $Z_{j,i}$.
6. Sprawdź, czy kolejka zadań jest pusta.
7. Jeśli nie, to idź do kroku 4. W przeciwnym razie idź do kroku 8.
8. Pobierz ilość pozostałych do obliczenia zadań (długości kolejek) z każdego procesora.
9. Wybierz maksymalną wartość.
10. Jeśli wynosi ona 0, to idź do kroku 14.
11. Podziel najdłuższą kolejkę zadań na 2 równe części.
12. Połowę zadań wstaw do własnej kolejki, a resztę zostaw.
13. Idź do kroku 4.

---

14. Połącz wygenerowane zbiory $\overline{S}_{j-1,p}$ z Q procesorów w zbiór $\overline{S}_{j-1}$.
15. $j = j - 1$.
16. Jeśli $j > 0$, to idź do kroku 2.
17. Koniec.

Przy pewnych uproszczeniach czas wykonania obliczeń na pojedynczym stopniu można oszacować jako:

$$T_j'' = \frac{T_j}{Q} + r_j \cdot \tau_m \tag{16}$$

gdzie:

r_j - liczba migracji zadań,

τ_m - czas wykonania migracji zadania.

Współczynnik przyspieszenia σ na stopniu j wynosi:

$$\sigma_j = \frac{T_j}{T_j''} = \frac{Q}{1 + Q \cdot \frac{r_j \cdot \tau_m}{T_j}} \tag{17}$$

Całkowity czas przetwarzania zadania Z wynosi:

$$T_c'' = \sum_{j=1}^{J} T_j'' = \frac{T_c}{Q} + r \cdot \tau_m \tag{18}$$

gdzie:

$$r = r_1 + r_2 + \ldots + r_J.$$

Jeśli iloczyn $r_j \cdot \tau_m$ jest mały w porównaniu z czasem T_j, to metoda pozwala uzyskać przyspieszenie Q-krotne.

4. Wyniki eksperymentów obliczeniowych

Przedstawione algorytmy zaimplementowano w postaci programu symulacyjnego działającego na pojedynczym komputerze jednoprocesorowym. Symulacje polegały na rzeczywistym wykonywaniu obliczeń z jednoczesną kontrolą, które fragmenty obliczeń wykonywałyby się na poszczególnych procesorach. Dla danego zadania obliczeniowego podaje się liczbę procesorów, dla jakiej ma być przeprowadzona symulacja.

Badania polegały na pomiarze czasu wykonywania obliczeń i porównaniu tych czasów dla obliczeń na pojedynczym procesorze, na kilku procesorach i na kilku procesorach z równoważeniem obciążenia. Dysponowano dwoma źródła zadań obliczeniowych:

- realne zadania opracowane w instytucie dla pewnych przypadków praktycznych,
- generowane zadania testowe.

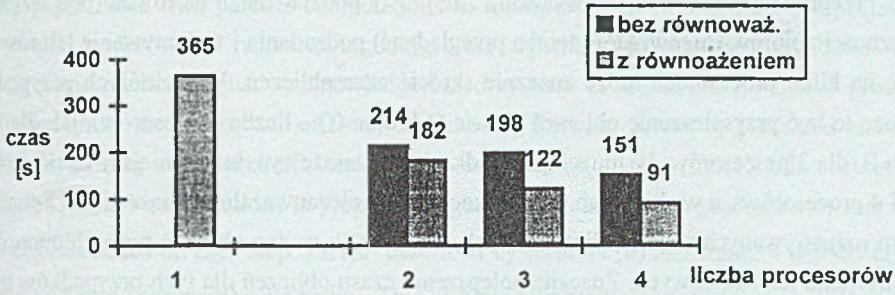
Poniżej przedstawiono wyniki eksperymentów symulacji dla dwóch przykładów zadań:

A - zadanie składające się z 21 faktów i 38 formuł elementarnych - realne,

B - zadanie składające się z 13 faktów i 37 formuł elementarnych - testowe.

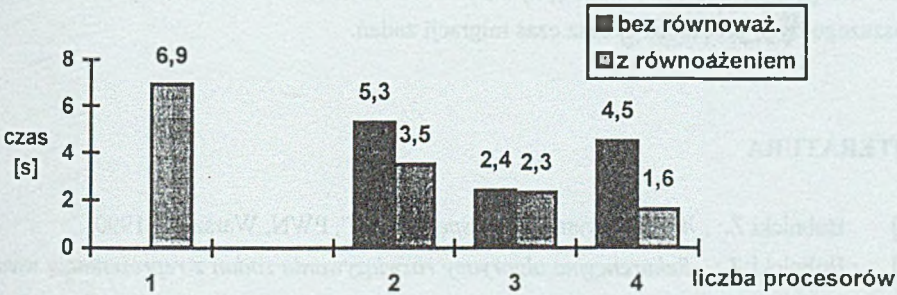
Wykresy przedstawiają czasy wykonywania zadania analizy na jednym i wielu procesorach. Dla symulacji wykonywania zadań na wielu procesorach podano czasy dla algorytmu bez równoważenia obciążenia i z dynamicznym równoważeniem obciążenia. We wszystkich przypadkach przyjęto, że czas podziału zadania na danym stopniu obliczeniowym na podzadania, czas przekazania podzadań do wykonania na poszczególne procesory i czas kompozycji zadania dla następnego stopnia obliczeń, oraz czas migracji zadań są pomijalnie małe.

1) Zadanie A



Rys. 6. Czasy wykonywania obliczeń na jednym i wielu procesorach dla zadania A
Fig. 6. The computation times on single- and multiprocessor for example A

2) Zadanie B.



Rys. 7. Czasy wykonywania obliczeń na jednym i wielu procesorach dla zadania B
Fig. 7. The computation times on single- and multiprocessor for example B

5. Podsumowanie

W artykule przedstawiono sposób realizacji rekurencyjnych algorytmów logiczno-algebraicznych w środowisku równoległego przetwarzania zadań. Zaprezentowano prosty sposób adaptacji oryginalnego algorytmu obliczeniowego dla metody logiczno-algebraicznej do systemów wieloprocesorowych. Taka prosta adaptacja może jednak prowadzić do nierównomiernego wykorzystania procesorów w systemie i w rezultacie do

zmniejszenia efektywności obliczeń. Dlatego też zaproponowano algorytm obliczeniowy dynamicznie równoważący obciążenie w systemie.

Przeprowadzone symulacje wykazały , że prosty podział zadań na równe (pod względem liczności zbiorów ciągów, które trzeba przeglądnąć) podzadania i wykonywanie ich równolegle na kilku procesorach może znacznie skrócić czas obliczeń. W niektórych przypadkach może to być przyspieszenie obliczeń prawie Q-krotne (Q - liczba procesorów), jak dla zadania B, dla 3 procesorów. W innych przypadkach zysk może być dużo mniejszy (zadanie B dla 2 i 4 procesorów), a w skrajnych przypadkach nawet niezauważalny. Wiąże się to z charakterem rozpatrywanych zadań obliczeniowych, dla których trudno określić pracochłonność wykonywania ich składowych. Znaczne polepszenie czasu obliczeń dla tych przypadków umożliwia zastosowanie zaprezentowanego algorytmu dynamicznego równoważenia obciążenia z migracją zadań, co również wykazały przeprowadzone eksperymenty obliczeniowe.

Należy zauważyć, że przedstawione wyniki nie uwzględniają czasów komunikacji między zadaniami. Odpowiadają one warunkom obliczeń na komputerach wieloprocesorowych. W przypadku obliczeń w sieci komputerowej należy dodatkowo uwzględnić następujące elementy: czas przekazywania podzadań do wykonania na poszczególne procesory-komputery, czas kompozycji zadania na następny stopień obliczeń (czas zbierania wyników obliczeń z poszczególnych procesorów) oraz czas migracji zadań.

LITERATURA

[1] Bubnicki Z.: *„Wstęp do systemów ekspertowych"*, PWN, Warszawa 1990.

[2] Bubnicki Z.: *„Rekurencyjne algorytmy rozwiązywania zadań z reprezentacją wiedzy w systemie ekspertowym"*, Archiwum Informatyki Teoretycznej i Stosowanej, tom 3, 1991.

[3] Bubnicki Z.: *„Podstawy informatycznych systemów zarządzania"*, WPWR, Wrocław 1993.

[4] Borzemski L., Fraś M. i inni: *„Systemy ekspertowe CONTROL-LOG/S i CLASS-LOG/S."*, III Krajowa Konferencja KOWBAN - Materiały, Polanica Zdrój 1996.

[5] Fraś M.: *„Metody przyspieszenia obliczeń w algorytmach logiczno-algebraicznych"*, IV Krajowa Konferencja KOWBAN - Materiały, Świeradów Zdrój 1997.

Recenzent: Dr inż. Marcin Skowronek

Wpłynęło do Redakcji 15 grudnia 1997 r.

Abstract

The paper deals with parallel computations in multiprocessor environment - especially with parallel execution of recursive logic-algebraic algorithm. In section 1 the basics of logic-algebraic method applied to knowledge-based systems is presented. Recursive procedure (based on the decomposition of facts) described by formula (3) can be presented in uniprocessor system as a sequence of J tasks being computed on J steps (fig. 1,2). In section 2 the idea of parallel computations is proposed. Figure 3 depicts parallel generation of sets of sequences calculated on each step $j \in \overline{1,J}$ described by formula (6) and figure 4 depicts task execution on step j for parallel algorithm. This original algorithm can lead to unequal load of processors in multiprocessor hence dynamic load balancing algorithm is proposed in section 3. The results of computational experiments for two examples are presented in section 4. Computation times for systems with 1, 2, 3 and 4 processors are presented on figure 6 (example A) and figure 7 (example B). There are compared computation times for algorithm with (dark grey column) and without load balancing (light grey column).