ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1995
Seria: INFORMATYKA z. 30 Nr kol. 1222

Wojciech MIKANIK

TRANSPUTEROWA REALIZACJA RÓWNOLEGŁEGO ALGORYTMU PRZESZUKIWANIA WYCZERPUJĄCEGO

Streszczenie. Minimalną doskonałą funkcją mieszającą (MDFM) nazywamy injekcję ze zbioru kluczy W w przedział liczb całkowitych $[0..card(W)-1]$. Majewski i Czech (*The Computer Journal* 36, 6 (1993)) przedstawili algorytm znajdowania MDFM o postaci $F(w) = (h_0(w)+g\circ h_1(w)+g\circ h_2(w)) \bmod card(W), w \in W$. Jeden z kroków algorytmu— przeszukiwanie wyczerpujące — ma wykładniczą pesymistyczną złożoność czasową. W niniejszym artykule przedstawiono równoległy algorytm przeszukiwania wyczerpującego. Badania eksperymentalne z jego implementacją dla transputerów pokazują, że skraca on czas generowania MDFM.

TRANSPUTER IMPLEMENTATION OF PARALLEL EXHAUSTIVE SEARCH

Summary. Minimal perfect hash function (MPHF) is an injection from a set of keys W into an interval of integers $[0..card(W)-1]$. Majewski and Czech (*The Computer Journal* 36, 6 (1993)) presented an algorithm for finding MPHF of the form $F(w) = (h_0(w) + g \circ h_1(w) + g \circ h_2(w)) \bmod card(W), w \in W$. One of the steps of the algorithm — exhaustive search — has an exponential worst-case time complexity. In this article a parallel algorithm for the exhaustive search is presented. The experiments with an implementation of the algorithm for transputers show that it reduces the time of finding MPHF.

RÉALISATION D'UN ALGORITHME PARALLÈLE DE RECHERCHE EXHAUSTIVE SUR TRANSPUTERS

Résumé. La fonction minimale et parfaite de dispersion (FMPD) est une injection de l'ensemble des clés W sur l'intervalle des entiers naturels $[0..card(W)-1]$. Majewski et Czech (*The Computer Journal* 36, 6 (1993)) ont présenté un algorithme de recherche FMPD de forme $F(w) = (h_0(w) + g \circ h_1(w) + g \circ h_(w)) \bmod card(W), w \in W$. Une de ses étapes — la recherche exhaustive — a une exponentielle complexité pessimiste de temps. Dans cet article on a présenté un algorithme parallèle de recherche exhaustive. Les expériences effectués avec une implémentation sur transputers ont montré que cet algorithme raccourci le temps de recherche de la FMPD.

1. Wstęp

Doskonałą funkcją mieszającą (DFM) nazywamy injekcję F ze zbioru kluczy W w przedział liczb całkowitych $[0..N-1]$, gdzie $N \geq card(W)$. Jeśli $N = card(W)$, bijekcję F nazywamy minimalną, doskonałą funkcją mieszającą (MDFM). Z pomocą takiej funkcji można dla danego zbioru kluczy w jednej próbie odnajdywać poszukiwany klucz w tablicy bądź stwierdzać, że klucz taki w tablicy nie występuje. Metodę tę stosuje się w różnego rodzaju programach tłumaczących, np. w kompilatorach, interpretatorach, asemblerach, w słownikach komputerowych oraz wszędzie tam, gdzie zachodzi potrzeba szybkiego wyszukiwania informacji, a zbiór kluczy znany jest na początku tworzenia systemu i nie ulega zmianie w trakcie jego pracy.

Znane są rozmaite algorytmy znajdowania MDFM. Jeden z nich — o liniowej złożoności czasowej — przedstawili Czech i Majewski w pracy [2]. Reprezentacja MDFM generowana tym algorytmem wymaga $aN + O(1)$ słów pamięci o długości $\log_2 N$ bitów każde, gdzie $a = 2$. Zmniejszanie wartości a powoduje znaczne wydłużenie czasu realizacji algorytmu, ze względu na wykonywanie przeszukiwania wyczerpującego, które ma wykładniczą złożoność czasową. W niniejszym artykule zaprezentowano równoległy algorytm przeszukiwania wyczerpującego, który przez wykorzystanie systemu wieloprocesorowego pozwala na skrócenie czasu generowania MDFM dla małych wartości a.

W punkcie 2 przedstawiono krótki opis algorytmu Czecha i Majewskiego. Punkt 3 zawiera omówienie równoległego algorytmu przeszukiwania wyczerpującego. W punkcie 4 opisano badania eksperymentalne przeprowadzone z wykorzystaniem transputerowej implementacji równoległego algorytmu. W punkcie 5 zaprezentowano uzyskane wyniki oraz ich analizę teoretyczną. Podsumowanie zamieszczono w punkcie 6.

2. Algorytm Czecha i Majewskiego generowania MDFM

Omawiany algorytm znajduje minimalną doskonałą funkcję mieszającą postaci $F(w) = (h_0(w) + g \circ h_1(w) + g \circ h_2(w)) \bmod card(W)$, gdzie $w \in W$, h_0, h_1, h_2 są pomocniczymi funkcjami pseudolosowymi, g — funkcją wyznaczaną w trakcie realizacji algorytmu, która reprezentowana jest przy pomocy tablicy o rozmiarze $2a \times card(W)$. Algorytm składa się z kilku kroków. Przy jego omawianiu przyjęto konwencję, iż pary i trójki nieuporządkowane zapisywane są z pomocą nawiasów okrągłych np. (a, b, c), zaś pary i trójki uporządkowane - z pomocą nawiasów kątowych np. $\langle a, b, c\rangle$.

W pierwszym kroku wyznaczane są funkcje h_0, h_1, h_2 takie, że $\forall w_1, w_2 \in W, w_1 \neq w_2 \Leftrightarrow \langle h_0(w_1), h_1(w_1), h_2(w_1)\rangle \neq \langle h_0(w_2), h_1(w_2), h_2(w_2)\rangle$. Wartości h_1, h_2 definiują graf zależności $G_0 = (V_0, E_0)$ taki, że $V_0 = \{h_1(w) : w \in W\} \bigcup \{h_2(w) : w \in W\}, E_0 = \{(h_1(w), h_2(w)) : w \in W\}$. Dla potrzeb dalszych rozważań przyjmijmy następujące oznaczenia: $W_f = \{w : (h_1(w), h_2(w))$ nie należy do żadnego cyklu w grafie $G_0\}$, $W' = W - W_f$ oraz $G = (V, E)$ taki, że $V = \{h_1(w) : w \in W'\} \bigcup \{h_2(w) : w \in W'\}, E = \{(h_1(w), h_2(w)) : w \in W'\}$.

W kroku drugim zbiór W' dzielony jest na podzbiory $W_0, W_1, \ldots, W_k$ takie, że $W_0 = \emptyset, W_i \subset W_{i+1}, W_i \neq W_{i+1}, i = 0, 1, \ldots, k-1, W_k = W'$. Ciąg $W_0, W_1, \ldots, W_k$ nazywany jest wieżą podzbiorów zbioru W'. Zbiór $X_i = W_i - W_{i-1}, i = 1, 2, \ldots, k$, jest określany mianem i–tego poziomu wieży. Niech $x_i \in X_i, i = 1, 2, \ldots, k$, oznacza klucz kanoniczny i–tego poziomu wieży. Jako klucz kanoniczny można wybrać dowolny element podzbioru X_i. Przyjmijmy także, że $Y_j = \{x_i : i = 1, 2, \ldots, j\}$ oraz $\forall i \in [1..k], \forall w \in W_i$ niech $s(w) = (y_0, y_1, \ldots, y_m)$ oznacza minimalny ciąg kluczy y_i taki, że ciąg krawędzi $((h_1(y_0), h_2(y_0)), (h_1(y_1), h_2(y_1)), \ldots, (h_1(y_m), h_2(y_m)))$ nad zbiorem wierzchołków V tworzy drogę od $h_1(w)$ do $h_2(w)$. Zachodzi wówczas $h_1(w) = h_1(y_0)$ oraz $h_2(w) = h_2(y_m)$.

W kolejnym kroku algorytmu szukane są takie wartości $U(x_i) \in [0..N-1], i = 1, 2, \ldots, k$, że dla funkcji $F(w)$ zdefiniowanej następująco:

$$F(w) = \begin{cases} (h_0(w) + U(w)) \bmod N; & w \text{ jest kluczem kanonicznym} \\ & w \in Y_i \\ (h_0(w) + \sum_{k=0}^{m} (-1)^k U(y_k)) \bmod N \;; & w \in W - W_f - Y_k \\ & \text{oraz } s(w) = (y_0, y_1, \ldots, y_m) \end{cases}$$

prawdziwa jest równoważność: $\forall w_1, w_2 \in W - W_f, w_1 = w_2 \Leftrightarrow F(w_1) = F(w_2)$. Krok ten jest realizowany algorytmem przeszukiwania wyczerpującego z powrotami. W algorytmie tym dla $i \in [1..k-1]$ wyznaczywszy $U(x_j)$ dla $j = 1, 2, \ldots, i$ (a więc także wartości $F(w), w \in W_i$) poszukuje się $U(x_{i+1})$ takiego, aby dla wszystkich $w \in W_{i+1}$ funkcja $F(w)$ była różnowartościowa. Jeśli takie $U(x_{i+1})$ nie zostanie znalezione wówczas wykonywany jest powrót i obliczenia są kontynuowane dla nowej wartości $U(x_i)$.

Po określeniu $F(w)$ dla kluczy $w \in W - W_f$, kluczom $w \in W_f$ przyporządkowuje się arbitralnie takie wartości $i \in [0..N-1]$, aby została zachowana różnowartościowość $F(w)$, a następnie znajdowana jest funkcja g reprezentowana w postaci tablicy taka, że $F(w) = (h_0(w) + g \circ h_1(w) + g \circ h_2(w)) \bmod N, w \in W$, jest minimalną doskonałą funkcją mieszającą. Bardziej szczegółowy opis algorytmu może czytelnik znaleźć w pracy [2].

3. Równoległy algorytm przeszukiwania wyczerpującego

W przedstawionym powyżej algorytmie sekwencyjnym przeszukiwanie wyczerpujące kończy się po znalezieniu pierwszego rozwiązania — minimalnej doskonałej funkcji mieszającej dla danego zbioru słów. Sprawdzany jest więc tylko pewien fragment drzewa przeszukiwań i tylko ten fragment powinien zostać przeszukany przez algorytm równoległy. Kale i Saletore [6] zaproponowali strategię nadawania rozwiązaniom częściowym priorytetów i przechowywanie ich w kolejce priorytetowej umieszczonej w pamięci współdzielonej dostępnej dla wszystkich procesorów przeszukujących. W przypadku architektury opartej na przesyle komunikatów naturalnym wyborem jest farma procesorów, opisana w pracach [5], [3]. Procesor *Nadzorca* rozdziela pracę między *Wykonawców*, realizujących przeszukiwanie wyczerpujące, oraz przechowuje uzyskane rozwiązania częściowe. W przedstawionym algorytmie generowania MDFM, rozwiązanie częściowe dla i poziomów wieży $W_0, W_1, \ldots, W_{i-1}$ zawiera dane określające wartości $U(x_j)$ dla $j = 0, 1, \ldots, i-1$. Początkowe fragmenty różnych rozwiązań częściowych mogą być identyczne. Użycie struktury danych zwanej *trie* (od ang. re*trie*val)[1] pozwala na wykorzystanie tego faktu w celu zmiejszenia rozmiaru pamięci koniecznej do ich przechowywania. Każdy węzeł *trie* zawiera następujące dane:

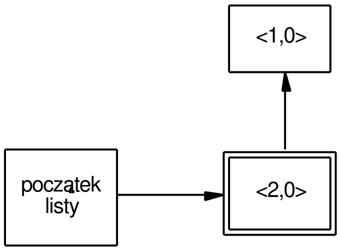
- numer poziomu wieży l;
- wartość parametru p zdefiniowanego następująco:

  $$p = (U(w_l) + h_0(w_l) - rnd\_h_0(w_l) + card(W)) \bmod card(W)$$

  gdzie w_l jest kluczem kanonicznym l–tego poziomu wieży, $rnd\_h_0(w_l)$ — pseudolosową liczbą całkowitą z przedziału $[0..card(W) - 1]$ przyporządkowaną kluczowi w_l;
- liczba synów węzła;
- znacznik określający czy poddrzewo, którego dany węzeł jest korzeniem, ma być przydzielone do przeszukania (na rysunkach węzły takie oznaczone są podwójną ramką);
- wskaźnik do ojca węzła;
- wskaźnik do jego lewego i prawego sąsiada.

Wszystkie liście *trie* tworzą dwukierunkową listę. Każdy z liści *trie* reprezentuje pojedyncze rozwiązanie częściowe. Nowe elementy są wstawiane do *trie* w taki sposób, aby ich uporządkowaie w liście odpowiadało kolejności, w jakiej poszczególne rozwiązania powinny być opracowywane. Jako pierwsze powinno być rozważane rozwiązanie częściowe reprezentowane przez skrajnie lewy element listy. Stan początkowy struktury *trie* pokazany jest na rys. 1.

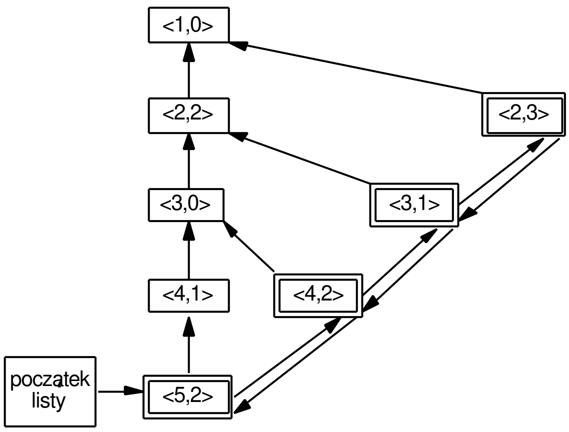
[1]Pomysł zastosowania tej struktury danych został zaczerpnięty z prac [1], [4], [7].



Rys. 1. Początkowy stan struktury *trie*
Fig. 1. The initial state of the *trie* structure

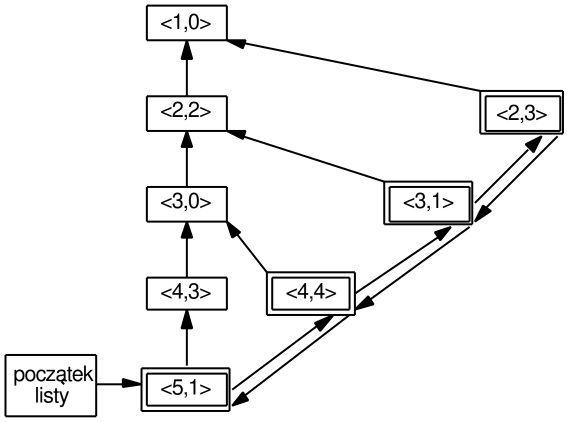
Węzły identyfikowane są uporządkowaną parą liczb, z których pierwsza jest numerem poziomu wieży, a druga — wartością p. Para $\langle 1,0\rangle$ znana jest każdemu procesorowi przeszukującemu w momencie rozpoczęcia obliczeń algorytmu. Przydzielenie pracy polega na wysłaniu numeru poziomu startowego, od którego rozpocznie się przeszukiwanie oraz wartości p dla tych poziomów, dla których jest to niezbędne. Warunkiem koniecznym przesłania danych przez *Wykonawcę* jest wystąpienie powrotu. Warunek wystarczający jest spełniony wtedy, gdy poziom aktualny jest wcześniejszy od startowego lub określono wartość $F(w)$ dla kluczy z co najmniej d poziomów wieży (licząc także poziom startowy). Wartość d jest parametrem algorytmu regulującym rozmiar ziarna.

Przykładowo, rozpatrzmy działanie algorytmu dla $d = 1$ i dwóch procesorów przeszukujących. W pierwszym kroku do jednego z nich wysyłane są dane o węźle $\langle 2,0\rangle$, jako że jest to jedyny liść. Załóżmy, że ulokowano słowa z poziomów $2,3,4,5$ z wartościami p równymi odpowiednio $2,0,1,1$, oraz że nie udało się znaleźć miejsca dla słów z poziomu 6. Rysunek 2 przedstawia stan *trie* dla tego rozwiązania częściowego. Następnie jednemu



Rys. 2. Stan *trie* dla pierwszego rozwiązania częściowego
Fig. 2. The state of the *trie* for the first partial solution

z procesorów przydzielone zostaje do przeszukania poddrzewo z korzeniem $\langle 5,2\rangle$, a drugiemu — z korzeniem $\langle 4,2\rangle$ (węzły z listy pobierane są od lewej strony). Przyjmijmy, że przeszukiwanie rozpoczęte od węzła $\langle 5,2\rangle$ zakończyło się porażką, a rozpoczęte od $\langle 4,2\rangle$ — ulokowaniem słów z poziomów 4 i 5 z wartościami p równymi odpowiednio 3 i 0. Rysunek 3 przedstawia stan *trie* po uwzględnieniu tych rezultatów. Realizacja algorytmu jest kontynuowana do momentu znalezienia rozwiązania lub sprawdzenia wszystkich moż-



Rys. 3. Stan *trie* dla kolejnych rezultatów częściowych
Fig. 3. The state of the *trie* for the next partial solutions

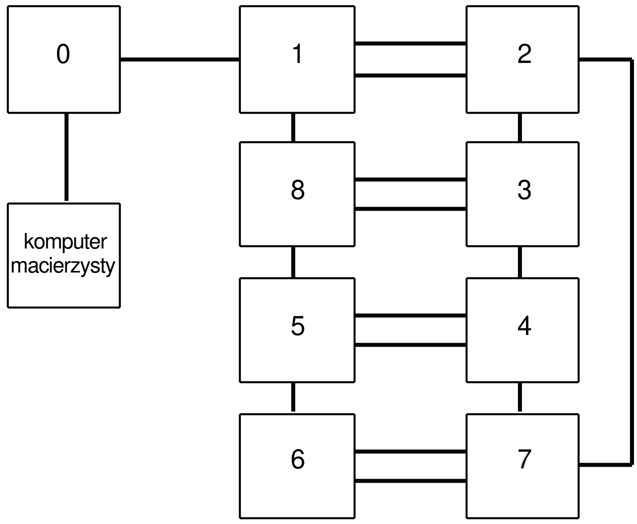
liwości.

4. Badania eksperymentalne

Algorytm zaimplementowano w języku Parallel C[2]. Badania eksperymentalne przeprowadzono w systemie dziewięciu transputerów T800-25, z których do obliczeń wykorzystano osiem. Dziewiąty transputer służył do realizacji operacji wejścia/wyjścia. Każdy z transputerów wyposażony był w 1 MB pamięci operacyjnej. Wszystkie transputery osadzone były na płycie IMS B008 umieszczonej jako karta rozszerzająca w komputerze klasy PC pracującym pod kontrolą systemu operacyjnego MS-DOS. Sposób połączenia transputerów na płycie B008 przedstawiono na rys. 4.

Eksperymenty przeprowadzono dla zbiorów zawierających $100, 200, \ldots, 1000$ słów wybranych w sposób losowy ze słownika języka angielskiego zawartego w systemie operacyjnym Unix. Ze względu na ograniczenia pamięciowe nie można było przeprowadzić badań dla liczniejszych zbiorów. Wykonano dwie serie pomiarów dla $a = 0.3$ oraz $a = 0.25$. W każdym eksperymencie generowano kilkakrotnie funkcję mieszającą dla tego samego zbioru słów stosując algorytm równoległy z wykorzystaniem kolejno $1, 2, 4, 7$ *Wykonawców* oraz stosując algorytm sekwencyjny. W tabeli 4. podano rozmieszczenie procesów roboczych w każdej z czterech prób. Zadanie rozdzielające pracę było zawsze wykonywane w transputerze o numerze 2. Transputer o numerze 0 służył do komunikacji z komuperem macierzystym. Czas generowania funkcji był ograniczony do 10 minut. Po jego upływie działanie programu było przerywane. Parametr $d = 1$ był stały.

[2]W trakcie prac korzystano z tekstu źródłowego programu w wersji sekwencyjnej opracowanego przez Bohdana Majewskiego z Uniwersytetu Queensland, Australia.



Rys. 4. Sposób połączenia używanych transputerów
Fig. 4. The connections between transputers

Tabela 4.
Numery procesorów wykonujących przeszukiwanie

Liczba procesorów przeszukujących	Numery transputerów
1	1
2	1, 3
4	1, 3, 7, 8
7	1, 3, 4, 5, 6, 7, 8

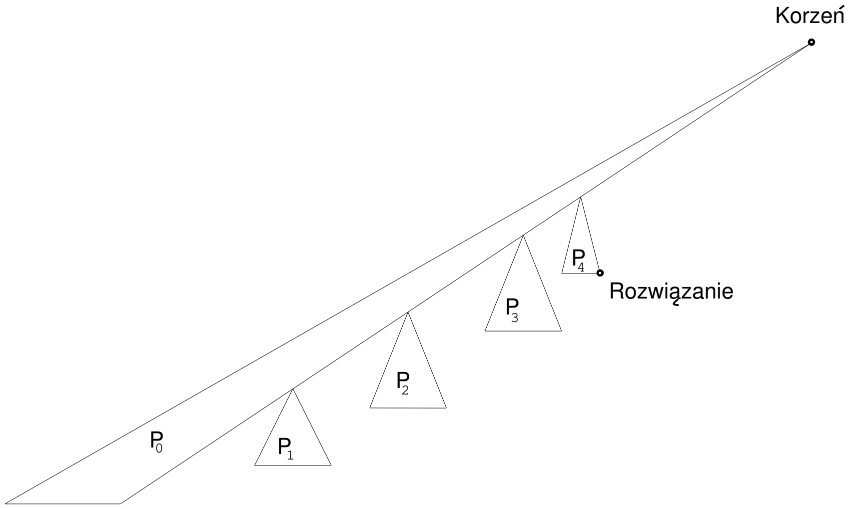
5. Wyniki eksperymentów

W analizie uwzględniono tylko te eksperymenty, w których funkcja mieszająca wygenerowana została w każdej próbie. Wartości średnie uzyskanych przyspieszeń przedstawiono w tabeli 5.

Tabela 5.
Wartości średnie
uzyskanych przyspieszeń

Liczba procesorów	Średnie przyspieszenie
2	19.2
4	21.0
7	26.1

Wysoka wartość uzyskanych średnich przyspieszeń wynika z faktu, iż w części eksperymentów uzyskano tzw. *superprzyspieszenia*, tj. przyspieszenia większe od liczby użytych procesorów. W celu wyjaśnienia tego zjawiska rozpatrzmy przykład zilustrowany na rys. 5, dla chwili gdy zakończyło się przeszukiwanie poddrzewa P_0 oraz dostępne są rezultaty częściowe, będące korzeniami poddrzew P_1, P_2, P_3 i P_4. Czasy przeszukiwania sekwencyj-



Rys. 5. Drzewo, dla którego $S_4 > 4$
Fig. 5. Tree, for which $S_4 > 4$

nego poddrzew $P_0, P_1, \ldots, P_4$ oznaczmy odpowiednio $T_0, T_1, \ldots, T_4$. Czas przeszukiwania poddrzewa P_0 w algorytmie równoległym przy użyciu n procesorów oznaczmy przez T_0^r. Poddrzewo P_4 zawiera poszukiwane rozwiązanie. W algorytmie równoległym poddrzewa P_1, P_2, P_3 oraz P_4 będą kolejno przydzielane *Wykonawcom* do przeszukania. W przypadku gdy $n > 3$ rozwiązanie zostanie znalezione w czasie T_4 od momentu zakończenia przeszukiwania poddrzewa P_0. Całkowity czas T_r poszukiwania rozwiązania w algorytmie równoległym równy jest

$$T_r = T_0^r + T_4. \tag{1}$$

W algorytmie sekwencyjnym rozwiązanie zostanie znalezione po przeszukaniu kolejno poddrzew $P_0, P_1, \ldots, P_4$, co zajmuje czas:

$$T_s = T_0 + \sum_{i=1}^{4} T_i. \tag{2}$$

Pomijając wpływ kosztów komunikacji między procesami, zależność między T_0^r a T_0 można wyrazić następująco:

$$nT_0^r = T_0 \tag{3}$$

a stąd $T_0^r = T_0$. Uwzględniając wzory (1), (2) i (3), przyspieszenie S_n w omawianym przypadku wyraża się wzorem

$$S_n = \frac{T_s}{T_r} = \frac{T_0 + \sum_{i=1}^{4} T_i}{\frac{T_0}{n} + T_4}. \tag{4}$$

Jeśli $T_0 \to 0$ to $S_n \to \frac{\sum_{i=1}^{4} T_i}{T_4}$ i nie zależy wprost od liczby użytych procesorów lecz od czasów T_1, T_2, T_3, T_4 przeszukiwania poddrzew P_1, P_2, P_3, P_4.

6. Podsumowanie

W niniejszym artykule zaprezentowano równoległy algorytm przeszukiwania wyczerpującego. Przedstawiono także wyniki badań eksperymentalnych przeprowadzonych z wykorzystaniem transputerowej implementacji tego algorytmu. Badania pokazały, że z pomocą algorytmu równoległego minimalna doskonała funkcja mieszająca może być znaleziona znacznie szybciej niż przy użyciu algorytmu sekwencyjnego. Zaobserwowano także uzyskiwanie superprzyspieszeń. W artykule dokonano analizy teoretycznej tego zjawiska.

LITERATURA

[1] Bartoszek B., Czech Z. J., Konopka M.: Parallel Searching for a First Solution, *Proc. 9th International Symposium on Computer and Information Sciences*, (7–9 November 1994), Antalya, Turkey, 132–139.

[2] Czech Z. J., Majewski B. S.: A Linear Time Algorithm for Finding Minimal Perfect Hash Functions, *The Computer Journal*, Vol. 36, No. 6, s. 579–587, (1993).

[3] Day W.: Farming: towards a rigorous definition and efficient transputer implementation, Transputer Systems — Ongoing Research, A.R. Allen, Ed., IOS Press, Amsterdam, 1992.

[4] Fredkin E.: Trie Memory, *Communications of the ACM*, Vol. 3, No. 9, s. 490–499, (1960).

[5] May D., Sheppard R.: Communicating Process Computers, Technical Report 22, INMOS 1987.

[6] Kale L. V., Saletore V. A.: Parallel State-Space Search for a First Solution with Consistent Linear Speedups, *International Journal of Parallel Programming*, Vol. 19, No. 4, s. 251–293, (1990)

[7] Knuth D. E.: The Art of Computer Programming, Vol. 3, Sorting and Searching, Addison–Wesley, Reading, Massachusetts 1973.

Abstract

Minimal perfect hash function (MPHF) is an injection from a set of keys W into an interval of integers $[0..card(W)-1]$. Majewski and Czech (*The Computer Journal* 36, 6 (1993)) presented an algorithm for finding MPHF of the form $F(w) = (h_0(w) + g \circ h_1(w) + g \circ h_2(w)) \bmod card(W), w \in W$. This algorithm is briefly described in Section 2. One of the steps of the algorithm — exhaustive search — has an exponential worst-case time complexity. In Section 3 a parallel algorithm for the exhaustive search is presented. A farm of processors is used. The processor *Master* distributes work among the *Workers* and holds partial solutions in a data structure called *trie*[3]. The initial state of the *trie* is presented in Fig. 1. The parallel algorithm was implemented in the Parallel C language. The experiments on a transputer system were carried out. The average speedups obtained are presented in Table 1. Their values are relatively high due to the superspeedups obtained in some experiments. In Fig. 5 a search tree is shown for which a speedup greater than the number of processors n $(n > 3)$ can be measured.

[3]The idea of employing this stucture was taken from [1], [4], [7].