ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1994

Seria: INFORMATYKA z. 27 Nr kol. 1259

Bożena BARTOSZEK

RÓWNOLEGŁY ALGORYTM PRZESZUKIWANIA DLA ZNAJDOWANIA MINIMALNEJ DOSKONAŁEJ FUNKCJI MIESZAJĄCEJ

Streszczenie. W pracy przedstawiono równoległy algorytm przeszukiwania wyczerpującego przestrzeni rozwiązań dla znalezienia pierwszego rozwiązania. Jest on częścią algorytmu konstruowania minimalnej doskonałej funkcji mieszającej. Jako model obliczeń równoległych przyjęto model MIMD z pamięcią rozproszoną. Opracowano nową, oryginalną strukturę danych, tzw. odwrócone drzewo wyszukiwań pozycyjnych, która umożliwia przeprowadzenie równoległego przeszukiwania. Algorytm został zaimplementowany dla sieci transputerów. Przeprowadzone eksperymenty pokazały, że algorytm wykazuje prawie liniowe przyspieszenie w funkcji liczby procesorów. Stwierdzono też, że zastosowana oryginalna struktura danych jest bardzo efektywna pod względem pamięciowym.

PARALLEL SEARCHING FOR FINDING MINIMAL PERFECT HASH FUNCTION

Summary. A parallel algorithm for conducting a search for a first solution to the problem of generating minimal perfect hash function is presented. A message-based distributed memory computer MIMD is assumed as a model of parallel computation. A data structure, called reversed trie (r-trie), was devised to carry out the search. The algorithm was implemented on a transputer network. The experiments showed that the algorithm exhibits a consistent and almost linear speedup as a function of the number of processors. The r-trie structure proved to be very highly memory efficient.

UN ALGORITHME PARALLÈLE DE RECHERCHE POUR OBTENIR LA FONTION MINIMALE ET PARFAITE DE DISPERSION

Résumé. Un algorithme parallèle menant la recherche de la première solution au problème de la génération de la fonction minimale et parfaite de dispersion est présenté. Le modèle de calcul parallèle MIMD à mémoire distribuée est supposé. Une nouvelle originale structure de données, l'arborescence inversé, a été conçue pour effectuer la recherche parallèle. L'algorithme a été implémenté sur un réseau de transputers. Les expériences ont prouvé que l'algorithme démontre une tendance d'accélération presque linéaire en fonction du nombre de processeurs. On a prouvé que l'arborescence inversé est très efficace en ce qui concerne l'utilisation de la mémoire.

1. Wprowadzenie

Niech W, $|W| = m$, będzie zbiorem słów o skończonej długości nad uporządkowanym alfabetem Σ. Funkcja mieszająca h zdefiniowana jako h: $W \rightarrow I$ odwzorowuje zbiór W w zadany przedział liczb całkowitych, powiedzmy $[0, n-1]$, gdzie $n \geq m$. Oznacza to, że funkcja mieszająca dla każdego słowa $k \in W$ wyznacza adres (liczbę całkowitą ze zbioru I), pod którym przechowywany jest rekord identyfikowany słowem k. Strukturę, która służy do przechowywania rekordów, nazywamy tablicą mieszającą. Słowa, dla których został wyznaczony ten sam adres w tablicy mieszającej, nazywamy synonimami. W przypadku istnienia synonimów mówimy również, że występują kolizje. W literaturze znane są liczne sposoby rozwiązywania kolizji [5, 6].

Jeżeli funkcja h jest różnowartościowa, to mówimy, że jest ona doskonałą funkcją mieszającą. Nie występuje wówczas kolizja, a każdy rekord może być odszukany w tablicy mieszającej w dokładnie jednej próbie. Jeżeli dla różnowartościowej funkcji h spełniony jest warunek $n = m$, to h jest *minimalną doskonałą funkcją mieszającą* (MDFM).

MDFM mają wiele zastosowań. Są używane np. do identyfikacji słów kluczowych języków programowania, do rozpoznawania zleceń w systemach operacyjnych przy wyszukiwaniu informacji w komputerowych słownikach języków naturalnych itd.

Istnieją różne algorytmy znajdowania MDFM. W wielu z nich stosowany jest algorytm przeszukiwania wyczerpującego, który kończy swoje działanie po znalezieniu pierwszego rozwiązania.

W niniejszej pracy przedstawiono równoległy algorytm przeszukiwania wyczerpującego w celu znalezienia pierwszego rozwiązania. Jako model obliczeń równoległych przyjęto model MIMD (ang. multiple instruction, multiple data stream) z pamięcią rozproszoną.

W punkcie drugim przedstawiono sekwencyjny algorytm znajdowania MDFM. Punkt 3 zawiera przegląd kilku równoległych algorytmów przeszukiwania. W punkcie 4 przedstawiono opis nowego równoległego algorytmu przeszukiwania oraz jego implementacji w języku occam. W punkcie 5 dokonano dyskusji wyników przeprowadzonych eksperymentów. Punkt 6 zawiera wnioski końcowe.

2. Sekwencyjny algorytm znajdowania MDFM

Czech i Majewski [1] podali algorytm znajdowania funkcji mieszających o liniowej złożoności czasowej. W zaproponowanym algorytmie dla każdego słowa w poszukiwana jest funkcja o postaci:

$$h(w)=(h_0(w)+g(h_1(w))+g(h_2(w))) \bmod m$$

gdzie h_o, h_1, h_2 to pomocnicze funkcje pseudolosowe, a g jest tablicą, której wartości wyznaczane są metodą przeszukiwania wyczerpującego. W celu wyznaczenia wartości funkcji h_o, h_1, h_2 zastosowano metodę Foxa [7], w której korzysta się z trzech tablic T_o, T_1 i T_2 złożonych z pseudolosowych liczb naturalnych. Dla danego słowa w będącego ciągiem znaków $w = a_1 a_2...a_{|w|}$, trójka wartości h_o, h_1, h_2 obliczana jest z użyciem następujących wzorów:

$$h_0(w)=(\sum_{i=1}^{|w|} T_0[i_m,a_i]) \bmod m$$

$$h_1(w)=(\sum_{i=1}^{|w|} T_1[i_m,a_i]) \bmod r$$

$$h_2(w)=((\sum_{i=1}^{|w|} T_2[i_m,a_i]) \bmod r)+r$$

gdzie $r = m$, $i_m=((i + |w|) \bmod |w|_{max}) + 1$, a $|w|_{max}$ jest maksymalną długością słowa.

Algorytm znajdowania MDFM składa się z trzech kroków: *odwzorowanie*, *porządkowanie* i *wyszukiwanie*. W pierwszym kroku - odwzorowania - każde słowo jest przekształcane w trójkę liczb pseudolosowych $h_0(w)$, $h_1(w)$ i $h_2(w)$ z użyciem tablic T_o, T_1 i T_2. W drugim kroku - porządkowania - dokonuje się podziału zbioru W na podzbiory W_o, W_1,..., W_l, takie że $W_0 = \emptyset$, $W_i \subset W_{i+1}$ i $W_l = W$. Ciąg utworzonych podzbiorów nazywamy *wieżą* o wysokości l, a każdy podziór $X_i = W_i - W_{i-1}$ to tzw. *poziom wieży*. W trzecim

kroku - wyszukiwania - przy użyciu metody wyszukiwania wyczerpującego wyznacza się adresy w tablicy mieszającej dla słów z kolejnych poziomów wieży.

Umieszczenie słowa w tablicy mieszającej wymaga określenia wartości $U(w) = g(h_1(w)) + g(h_2(w))$. Może istnieć ciąg słów $(w_0, w_1, ..., w_{j-1})$, taki że $h_1(w_i) = h_1(w_{i+1})$ oraz $h_2(w_i) = h_2(w_{(i+2) \bmod j})$ dla $i = 0, 2, 4, ..., j - 2$. Po przydzieleniu miejsc w tablicy mieszającej dla słów $w_0, w_1, ..., w_{j-2}$ ustalone są wartości $g(h_1(w_{j-1}))$ oraz $g(h_2(w_{j-1}))$. Tym samym pozycja słowa w_{j-1} w tablicy mieszającej jest zdeterminowana i wynosi:

$$h(w_{j-1}) = (h_0(w_{j-1}) + U(w_{j-1})) \bmod m$$

W danym ciągu słów, słowa $w_0, w_1, ..., w_{j-2}$ są niezależne, tzn. można dla nich wybrać dowolną pozycję w tablicy mieszającej. Słowa takie nazywamy *kanonicznymi*. Natomiast słowo w_{j-1} jest zależne, tzn. jego pozycja w tablicy mieszającej jest zdeterminowana. Słowo takie nazywamy *niekanonicznym*. Można łatwo zauważyć, że

$$U(w_{j-1}) = g(h_1(w_{j-1})) + g(h_2(w_{j-1})) = \sum_{p \in droga(w_j - 1)} (-1)^p U(w_p)$$

gdzie $droga(w_{j-1})$ jest ciągiem słów $(w_0, w_1, ..., w_{j-2})$. Tak więc

$$h(w_{j-1}) = (h_0(w_{j-1}) + \sum_{p \in droga(w_j - 1)} (-1)^p U(w_p)) \bmod m$$

Jeżeli pozycja $h(w_{j-1})$ w tablicy mieszającej jest zajęta, to powstaje kolizja i znalezienie MDFM dla tak określonych wartości g nie jest możliwe.

W trakcie przeszukiwania wyczerpującego rozwiązywany jest następujący problem kombinatoryczny: znaleźć $U(w_i) \in [0, m - 1]$, $i = 1, 2, ..., k$ (k jest wysokością wieży), takie że wartości $h(w_i) = (h_0(w_i) + U(w_i)) \bmod m$ dla słów kanonicznych $w_i \in X$ oraz wartości $h(w_j) = (h_0(w_j) + \sum_{p \in droga(w_j)} (-1)^p U(w_p)) \bmod m$ dla słów niekanonicznych $w_j \in X$ są różne. Oznacza to, że dla dowolnych słów w_1 i $w_2 \in W$ zachodzi $h(w_1) \neq h(w_2)$. Wartości $U(w_i)$ znajdowane są w trakcie przeszukiwania wyczerpującego na każdym poziomie X_i wieży. Przeszukiwanie rozpoczyna się od ulokowania słowa kanonicznego w_i w tablicy mieszającej. W tym celu poszukuje się wolnej pozycji w tablicy mieszającej dla słowa w_i. Poszukiwanie rozpoczyna się począwszy od pozycji $h_0(w_i)$, co odpowiada wartości $U(w_i)$ równej 0. Jeżeli pozycja ta jest zajęta, bada się kolejne miejsca w tablicy mieszającej modulo m (rozmiar tablicy). Po ulokowaniu słowa kanonicznego obliczana jest wartość $U(w_i)$ i możliwe jest wyznaczenie pozycji dla słów niekanonicznych na danym poziomie. Zbiór pozycji w tablicy mieszającej dla słów na poziomie X_i nazywamy *wzorcem*. Jeżeli wszystkie pozycje określone przez wzorzec nie są zajęte, to słowa zostają ulokowane w

tablicy i przetwarzany jest kolejny poziom wieży. W przeciwnym razie wzorzec jest przemieszczany "w górę" tablicy modulo m do momentu znalezienia nie zajętych miejsc. Z wyjątkiem pierwszego poziomu wieży przeszukiwanie to jest wykonywane w sytuacji, gdy tablica jest już częściowo wypełniona. Może się więc zdarzyć, że w tablicy nie zostanie znalezione miejsce dla wzorca odpowiadającego poziomowi X_i. W tej sytuacji konieczny jest powrót do wcześniejszych poziomów, dla których znalezione zostaną nowe wzorce i proces umieszczania słów kolejnych poziomów jest powtarzany. Jest to więc przeszukiwanie wyczerpujące z powrotami.

Problem znalezienia wartości $U(w_i)$ może mieć wiele rozwiązań. Jak już wspomniano, interesuje nas znalezienie tylko jednego rozwiązania.

Złożoność algorytmu przeszukiwania wyczerpującego zastosowanego w trzecim kroku algorytmu jest wykładnicza w funkcji liczby słów, które należy umieścić w tablicy. Czas realizacji przeszukiwania zależy również od rozmiaru tablicy g. Udowodniono, że jeżeli $|g| = 2m$, to czas wykonania przeszukiwania można zaniedbać. Zmniejszenie rozmiaru tablicy g powoduje, że czas realizacji przeszukiwania rośnie wykładniczo. Ponieważ tablica g definiuje MDFM, pożądane jest, by jej rozmiar był jak najmniejszy.

3. Równoległe algorytmy przeszukiwania

Przeszukiwanie wyczerpujące można rozpatrywać jako przeszukiwanie grafu stanoprzestrzennego zwanego *drzewem wyszukiwawczym*. Korzeniem drzewa jest wektor pusty. Jego synowie odpowiadają możliwym wyborom pierwszego stanu (współrzędnej a_1) w drzewie. Ogólnie wierzchołki k-tego poziomu odpowiadają wyborom k-tego stanu (współrzędnej a_k) przy ustalonym wyborze stanów $1, 2, ..., k-1$ (wektor $(a_1, a_2, ..., a_{k-1})$).

Najprostszym sposobem przeprowadzenia przeszukiwania w sposób równoległy jest przeszukanie różnych poddrzew drzewa wyszukiwawczego przez poszczególne procesy. W przypadku gdy jesteśmy zainteresowani znalezieniem jednego rozwiązania i rozwiązanie to zostanie znalezione w pierwszym poddrzewie, praca wykonana przez procesy przeszukujące pozostałe poddrzewa jest zmarnowana. W konsekwencji, stosując tę metodę możemy uzyskać przyspieszenie znacznie większe od liczby wykorzystanych procesorów lub spadek przyspieszenia ze wzrostem liczby procesorów. Lai i Sahni [3] jako pierwsi zaobserwowali i opisali wyżej wymienione anomalie przyspieszenia.

Kalé i Saletore [4] zaproponowali dwa kryteria oceny równoległych algorytmów przeszukiwania:

1. Czas potrzebny na znalezienie rozwiązania. Agorytm równoległy powinien zawsze znajdować rozwiązanie szybciej niż najlepszy algorytm sekwencyjny. Uzyskane przyspieszenie powinno być bliskie liczby użytych procesorów i powinno monotonicznie rosnąć wraz ze wzrostem ich liczby.

2. Rozmiar pamięci koniecznej do przeprowadzenia przeszukiwania. Rozmiar ten zależy od zastosowanej metody przeszukiwania. Wzrost rozmiaru pamięci może być liniowy aż do wykładniczego w funkcji wysokości drzewa.

W pracy [4] Kalé i Saletore zaproponowali metody równoległego przeszukiwania w głąb z użyciem priorytetów wierzchołków. Istotą metody jest to, by wierzchołki drzewa wyszukiwawczego były odwiedzane w takiej kolejności jak w algorytmie sekwencyjnym, tj. od strony lewej do prawej w drzewie. Przy takiej organizacji przeszukiwania praca wykonana na prawo od pierwszego rozwiązania jest zminimalizowana. W zaproponowanej metodzie stosuje się wektory bitów określające priorytety wierzchołków. Priorytety, porównywane leksykograficznie, są przypisywane wierzchołkom w sposób dynamiczny w momencie tworzenia wierzchołków. Korzeniowi drzewa wyszukiwawczego przypisuje się wektor o długości zerowej. Priorytet syna danego wierzchołka otrzymuje się przez dopisanie numeru syna, wyznaczonego przez uporządkowanie wszystkich synów od strony lewej do prawej, do priorytetu wierzchołka będącego ojcem. Wierzchołki aktywne są przechowywane w kolejce priorytetowej współdzielonej przez wszystkie procesy. Wyznacza ona kolejność, w jakiej zostaną przeszukane wierzchołki drzewa.

Długość kolejki w najgorszym przypadku jest $O(pdb)$. W celu zmniejszenia długości kolejki Kalé i Saletore proponują zastosowanie dwóch technik zwanych: *binarną dekompozycją* i *opóźnionym uwalnianiem*. Ze względu na to, że technika zwana opóźnionym uwalnianiem została zastosowana w opracowanym algorytmie, omówimy krótko jej ideę. Technika opóźnionego uwalniania polega na tym, że w trakcie przeszukiwania synowie odwiedzanego wierzchołka nie są od razu dostępni dla pozostałych procesów, a jedynie dla procesu, który wygenerował tych synów. Postępuje się tak do momentu napotkania liścia w drzewie wyszukiwawczym. Wtedy wszystkie wierzchołki zostają uwolnione i mogą być pobrane przez pozostałe procesy. W wyniku takiego postępowania procesy pomijają pośrednie poziomy drzewa i przeszukują drzewo "od spodu".

Autorzy [4] twierdzą, że dzięki temu praca marnowana jest praktycznie wyeliminowana, a rozmiar wymaganej pamięci jest $O(p + d)$.

4. Równoległy algorytm przeszukiwania wyczerpującego

Przeszukiwanie wyczerpujące można rozpatrywać jako przeszukiwanie drzewa wyszukiwawczego, którego wierzchołki przechowują wartości U dla poszczególnych poziomów wieży. Droga od korzenia drzewa do wierzchołka na poziomie i wyznacza pewne rozwiązanie częściowe $U(w_1)$, $U(w_2)$,..., $U(w_i)$, zawierające wartości U dla słów kanonicznych. Wartości te określają pozycje w tablicy mieszającej dla słów na poziomach X_1, X_2,..., X_i.

W prezentowanym algorytmie zastosowano technikę *farmy procesorów*. W technice tej występuje centralny kontroler - proces *Nadzorca*, który przydziela pracę pewnej liczbie procesów typu *Wykonawca*. Proces *Nadzorca* przechowuje aktywne wierzchołki drzewa wyszukiwawczego reprezentujące rozwiązania częściowe problemu znajdowania MDFM. Rozwiązanie jest określone przez sekwencję wartości U. Pojedyncza wartość U z tej sekwencji odpowiada jednemu poziomowi wieży. Wierzchołki przesyłane są do procesów *Wykonawca*, które lokują kolejne poziomy wieży w tablicy mieszającej i wysyłają wyniki swojej pracy do procesu *Nadzorca*.

Rozwiązania częściowe są przechowywane w oryginalnej strukturze danych, którą nazwano *odwróconym drzewem wyszukiwań pozycyjnych* (ODWP). Struktura ta jest wariantem drzewa wyszukiwań pozycyjnych przedstawionym przez Knutha w pracy [5], w której wskaźniki skierowane są w stronę ojców, a nie synów wierzchołków. ODWP jest drzewem, którego wierzchołki odpowiadają wartościom U. Wierzchołki na poziomie l reprezentują zbiór rozwiązań częściowych dla ustalonej wcześniej sekwencji wartości U. Pojedynczy wierzchołek drzewa przechowuje: numer poziomu, wartość U dla tego poziomu, znacznik wskazujący, czy wierzchołek jest aktywny oraz wskaźnik do ojca wierzchołka. Pojedynczy wierzchołek ODWP jest identyfikowany przez parę (numer poziomu, wartość U). Wierzchołek (1, 0) jest korzeniem drzewa. Częściowe lub kompletne rozwiązanie można odtworzyć przechodząc od liścia drzewa do jego korzenia. Wierzchołki aktywne w drzewie są połączone w dwukierunkową listę, z której proces *Nadzorca* pobiera je i przydziela *Wykonawcom*.

Przykład 1. Rys. 1b pokazuje przykładowe ODWP. Wierzchołki na poziomie 3 reprezentują rozwiązanie częściowe, które określają następujące sekwencje wartości U: (0, 5, 2) i (0, 5, 3).

Dzięki zastosowaniu nowej struktury nie musimy wyznaczać priorytetów bitowych dla wierzchołków jak w rozwiązaniu Kalé i Saletore. Zbędne jest również tworzenie kolejki priorytetowej dla wierzchołków. W celu odnalezienia pierwszego od strony lewej w drze-

wie aktywnego wierzchołka należy w najgorszym przypadku odwiedzić co najwyżej $p - 1$ wierzchołków.

4.1. Proces *Wykonawca*

Wykonawca odbiera trzy rodzaje wiadomości od *Nadzorcy*:

1. *Nowe*. Wiadomość ta zawiera nowe częściowe rozwiązanie (ciąg numerów poziomów wraz z odpowiadającymi im wartościami U), od którego *Wykonawca* powinien kontynuować przeszukiwanie.

2. *Kontynuuj*. Wiadomość *Kontynuuj* nie zawiera składowych. Po jej otrzymaniu *Wykonawca* kontynuuje przeszukiwanie z ostatnio znalezionym rozwiązaniem częściowym.

3. *Koniec*. Wiadomość *Koniec* nie zawiera składowych. Jest wysyłana przez *Nadzorcę* wtedy, gdy zostało znalezione rozwiązanie lub stwierdzono, że nie ma rozwiązania.

Załóżmy, że *Wykonawca* ulokował ostatnio poziom o numerze i i odebrał wiadomość *Nowe*, która zawiera rozwiązanie częściowe obejmujące poziomy $j, j + 1, ..., n$. *Wykonawca* zwalnia w tablicy mieszającej pozycje zajęte przez słowa z poziomów $j, j + 1, ..., i$, a następnie umieszcza w tablicy słowa z poziomów $j, j + 1, ..., n - 1$. Pozycje tych słów w tablicy mieszającej są wyznaczone przez odebrane wartości U. *Wykonawca* kontynuuje przeszukiwanie od poziomu n z odebraną wartością U dla tego poziomu.

W prezentowanym algorytmie zastosowano technikę opóźnionego uwalniania. W konsekwencji, umieszczanie kolejnych poziomów wieży w tablicy mieszającej jest kontynuowane do momentu, gdy *Wykonawca* napotka w drzewie wyszukiwawczym liść, tj. poziom, który nie może być ulokowany w tablicy z danym rozwiązaniem częściowym. Załóżmy, że *Wykonawca* wykonuje powrót do poziomu k. Jeżeli $k < n$, to *Wykonawca* wysyła do *Nadzorcy* wiadomość *Powrót*. $k \geq n$ oznacza, że *Wykonawca* umieścił w tablicy mieszającej pewną liczbę poziomów wieży. Minimalna liczba poziomów, jaką *Wykonawca* musi ulokować w tablicy, zanim skomunikuje się z *Nadzorcą*, jest parametrem algorytmu. Wartość tego parametru można rozpatrywać jako *rozmiar ziarna* lub *granulację pracy*. Oznaczmy ten parametr jako s. Jeżeli $k \geq n + s$, to *Wykonawca* ulokował wymaganą liczbę poziomów i wysyła do *Nadzorcy* wiadomość *Sukces* (wartości U dla ulokowanych poziomów wieży). W przeciwnym razie *Wykonawca* kontynuuje przeszukiwanie.

W celu przyspieszenia początkowej fazy przeszukiwania każdy *Wykonawca* po ulokowaniu pierwszego poziomu wieży z $U[1] = 0$, kontynuuje przeszukiwanie do momentu

wystąpienia pierwszego powrotu. Wtedy pierwszy *Wykonawca* wysyła do *Nadzorcy* wiadomość o ulokowanych poziomach, a następnie każdy *Wykonawca* cofa się o liczbę poziomów równą jego numerowi, tzn. pierwszy *Wykonawca* cofa się o jeden poziom, drugi *Wykonawca* o dwa itd. Jeżeli *Wykonawca* wykonuje powrót poniżej drugiego poziomu, to zwalnia w tablicy mieszającej wszystkie pozycje z wyjątkiem pozycji zajętych przez pierwszy poziom i czeka na wiadomość *Nowe* od *Nadzorcy*. W przeciwnym razie *Wykonawca* kontynuuje przeszukiwanie od poziomu, do którego wykonał powrót.

4.2. Proces *Nadzorca*

Proces *Nadzorca* przechowuje rozwiązania częściowe i przydziela aktywne wierzchołki *Wykonawcom*. Oprócz tego przechowuje wskaźniki do wierzchołków ostatnio przetwarzanych przez każdego z *Wykonawców* i tworzy kolejkę numerów bezczynnych *Wykonawców*. Kiedy do drzewa zostaną dołączone nowe aktywne wierzchołki, *Nadzorca* wysyła je bezczynnym *Wykonawcom*.

Proces *Nadzorca* otrzymuje dwa rodzaje wiadomości od *Wykonawców*:

1. *Sukces*. Wiadomość ta zawiera numery ulokowanych poziomów wraz z odpowiadającymi im wartościami U. Załóżmy, że *Wykonawca* ulokował n poziomów zaczynając od poziomu i (wykonał powrót na poziomie $i + n$ do poziomu $i + n - 1$). Oznaczmy otrzymane wartości U jako $U[i]$, $U[i + 1]$,..., $U[i + n - 1]$. Jeżeli ostatnio przetwarzany przez *Wykonawcę* wierzchołek był pierwszym na liście wierzchołków aktywnych, proces *Nadzorca* wysyła do *Wykonawcy* wiadomość *Kontynuuj*. Następnie *Nadzorca* zmienia wartość U wierzchołka ostatnio przetwarzanego przez *Wykonawcę* na $U[i]$ i wstawia do drzewa wierzchołki $(i + 1, U[i + 1])$, $(i + 2, U[i + 2])$,..., $(i + n - 1, U[i + n - 1])$. Oprócz tego *Nadzorca* wstawia do drzewa wierzchołki aktywne $(i, U[i] + 1)$, $(i + 1, U[i + 1] + 1)$,..., $(i + n - 2, U[i + n - 2] + 1)$, które stają się wierzchołkami alternatywnymi dla dalszego procesu przeszukiwania. Jeżeli *Wykonawca* nie przetwarzał pierwszego wierzchołka w liście wierzchołków aktywnych, to *Nadzorca* znajduje taki wierzchołek i wyznacza rozwiązanie częściowe odpowiadające temu wierzchołkowi. Rozwiązanie to jest porównywane z rozwiązaniem częściowym otrzymanym od *Wykonawcy*, a fragment, w którym rozwiązania te się różnią, jest wysyłany do procesu *Wykonawca* jako wiadomość *Nowe*.

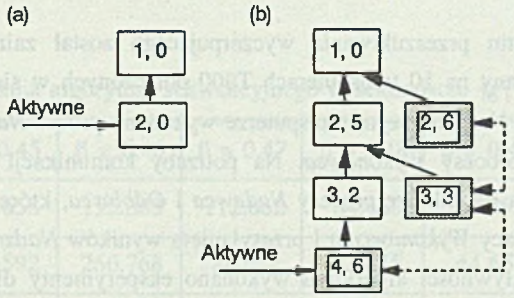
2. *Powrót*. Wiadomość ta nie zawiera składowych. Jest wysyłana wtedy, gdy proces *Wykonawca* cofnął się do poziomu o jeden niższy niż poziom, od którego zaczął. Załóżmy, że jest to poziom i, a ostatnio przetwarzany przez *Wykonawcę* wierzchołek oznaczmy jako $(j, U[j])$. *Nadzorca*, począwszy od wierzchołka $(j, U[j])$, schodzi w dół drzewa aż do napo-

tkania wierzchołka (i, $U[i]$). Następnie usuwa z drzewa wszystkie aktywne wierzchołki będące potomkami wierzchołka (i, $U[i]$) i wysyła do *Wykonawcy* wiadomość *Nowe*.

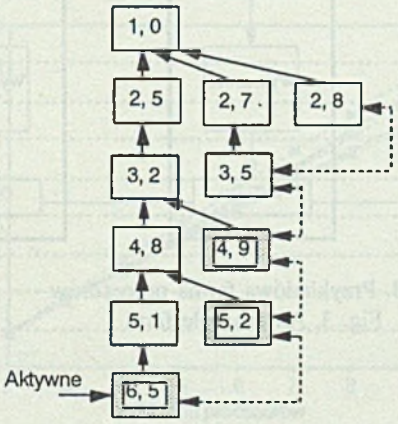
Przykład 2. Załóżmy, że przeszukiwanie jest wykonywane przez trzy procesy *Wykonawca*: W_1, W_2 i W_3. Początkowo w drzewie znajdują się trzy wierzchołki: (1, 0) i (2, 0) - rys. 1a. Załóżmy, że w trakcie przeszukiwania *Nadzorca* otrzymał następujące wiadomości:

Numer wiadomości	Typ wiadomości	Nadawca	Wartości U
1	*Sukces*	W_1	(5, 2, 6)
2	*Sukces*	W_1	(8, 1, 5)
3	*Powrót*	W_2	–
4	*Sukces*	W_3	(7, 5)

W odpowiedzi na wiadomość 1 *Nadawca* zmienia wartość U wierzchołka (2, 0) na 5 i umieszcza w drzewie następujące wierzchołki: (3, 2), (4, 6), (2, 6) i (3, 3). Wierzchołki (4, 6), (3, 3) i (2, 6) stają się wierzchołkami aktywnymi (rys. 1b). Jak już wyżej wspomniano, proces W_1 cofa się o jeden poziom i kontynuuje przeszukiwanie od wierzchołka (4, 6). Proces W_2 cofa się o dwa poziomy i kontynuuje przeszukiwanie od wierzchołka (3, 3), a proces W_3 od wierzchołka (2, 6). Załóżmy, że proces W_1 ulokował kolejne trzy poziomy (wiadomość 2). Ponieważ wierzchołek (4, 6) jest pierwszym wierzchołkiem na liście wierzchołków aktywnych, *Nadzorca* wysyła do W_1 wiadomość *Kontynuuj*, a następnie zmienia wartość U wierzchołka (4, 6) na 8. Do drzewa wstawione zostają aktywne wierzchołki (5, 1), (6, 5), (4, 9) i (5, 2). Załóżmy teraz, że W_2 wysyła wiadomość 3. Nadzorca po otrzymaniu tej wiadomości usuwa z drzewa wierzchołek (3, 3), po czym wysyła do W_2 wiadomość *Nowe* z rozwiązaniem częściowym (3, 2), (4, 8), (5, 2). Proces W_2 będzie kontynuował przeszukiwanie od poziomu numer 5. Załóżmy, że proces W_3 ulokował kolejne dwa poziomy (wiadomość 4). *Nadzorca* zmienia wartość U wierzchołka (2, 6) na 7 i wstawia do drzewa wierzchołki (3, 5) i (2, 8). Pierwszym wierzchołkiem na liście wierzchołków aktywnych i nie przydzielonych żadnemu procesowi jest wierzchołek (4, 9). Tak więc *Nadzorca* wysyła do procesu W_3 nowe częściowe rozwiązanie (2, 5), (3, 2), (4, 9). Proces W_3 będzie kontynuował przeszukiwanie od poziomu numer 4. Rys. 2 pokazuje stan drzewa po przetworzeniu wiadomości 4. Wierzchołki (6, 5), (5, 2) i (4, 9) są przydzielone procesom W_1, W_2, W_3.



Rys. 1(a) Początkowy stan drzewa; (b) Stan drzewa po przetworzeniu wiadomości 1
Fig. 1(a) Initial state of the r-trie; (b) The r-trie after message 1 has been processed

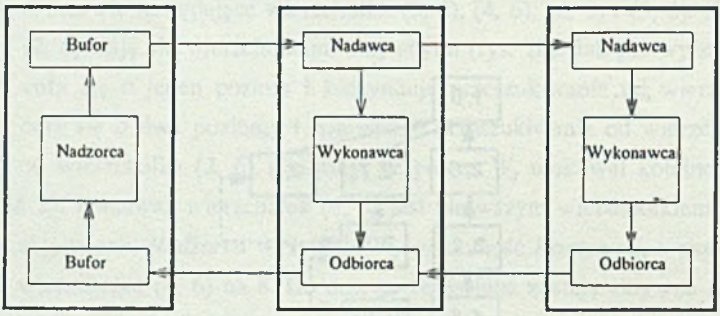


Rys. 2. Stan drzewa po przetworzeniu wiadomości 4
Fig. 2. The r-trie after message 4 has been processed

5. Wyniki eksperymentów

Równoległy algorytm przeszukiwania wyczerpującego został zaimplementowany w języku occam i wykonany na 10 transputerach T800 połączonych w sieć liniową w komputerze Meiko Surface. Na pierwszym transputerze wykonano proces *Nadzorca*, a na pozostałych transputerach procesy *Wykonawca*. Na potrzeby komunikacji międzyprocesowej zaimplementowano dwa dodatkowe procesy *Nadawca* i *Odbiorca*, które zajmują się odpowiednio rozsyłaniem pracy *Wykonawcom* i przesyłaniem wyników *Nadzorcy* (rys. 3).

W celu oceny efektywności algorytmu wykonano eksperymenty dla sześciu zbiorów danych zawierających od $m = 50$ do $m = 100$ słów. W Tablicy 1 podano czasy wykonania sekwencyjnego algorytmu przeszukiwania wykonanego na jednym transputerze dla tych danych. Na Rys. 3 pokazano wykres średniej wartości uzyskanego przyspieszenia w funkcji liczby procesorów. Granulacja pracy s jest parametrem wykresu. Każdy punkt wykresu jest wartością średnią z 36 wyników otrzymanych dla 6 zbiorów danych i wartości parametru $\beta = 0.45, 0.46, ..., 0.5$.



Rys. 3. Przykładowa farma procesorów
Fig. 3. An example farm

6. Podsumowanie

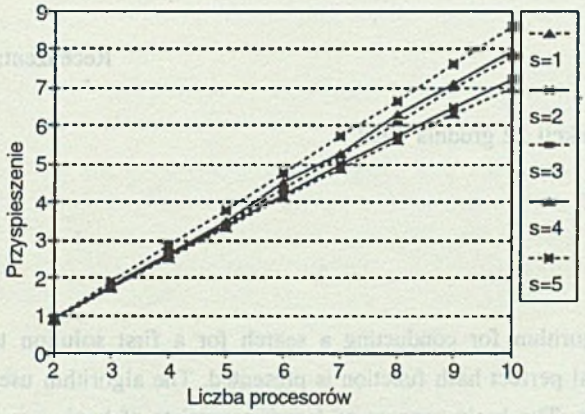
W pracy przedstawiono równoległy algorytm przeszukiwania wyczerpującego dla znajdowania minimalnej doskonałej funkcji mieszającej. W algorytmie zastosowano technikę farmy procesorów i opracowano nową oryginalną strukturę danych, tzw. odwrócone drzewo wyszukiwań pozycyjnych. Wyniki przeprowadzonych eksperymentów pokazały, że równoległy algorytm wykazuje prawie liniowe przyspieszenie w funkcji liczby procesorów

(rys. 4). Zastosowana struktura danych jest bardzo efektywna pod względem pamięciowym.

Tabela 1

Czasy wykonania algorytmu sekwencyjnego w sekundach, $|g| = \beta m$

m	β = 0.45	β = 0.46	β = 0.47	β = 0.48	β = 0.49	β = 0.50
50	227.658	195.883	112.862	84.468	58.293	21.217
60	382.592	260.768	80.682	73.845	44.657	10.769
70	385.010	250.005	144.504	66.800	57.800	13.147
80	318.291	200.772	205.961	45.931	29.858	17.782
90	402.634	322.956	181.241	123.641	55.215	14.851
100	865.847	445.430	244.551	115.510	57.919	44.709



Rys. 4. Przyspieszenie w funkcji liczby procesorów
Fig. 4. Speedup versus number of processors

LITERATURA

[1] Czech, Z.J., Majewski B.S.: A linear time algorithm for finding minimal perfect hash function, *The Computer Journal*, 1993, **36**.

[2] Bartoszek, B., Czech, Z.J., Konopka, M.: Parallel searching for a first solution, Tech. Rep. 8-93, University of Kent, 1993.

[3] Lai, T.H., Sahni, S.: Anomalies in parallel branch-and-bound algorithms, *Comm. ACM*, 1984, **6**, 594-602.

[4] Kalé, L.V., Saletore, A.: Parallel state-space search for a first solution with consistent linear speedups, *International Journal of Parallel Programming*, 1987, **19**, 251-293.

[5] Knuth, D.E.: *The Art of Computer Programming*, Vol. 3, *Sorting and Searching*, (Addison-Wesley, New York, 1973) 481-499.

[6] Lewis T.G., Cook C.R.: Hashing for dynamic and static internal tables, Computer, 1988, 45-56.

[7] Fox E.A., Chen Q.F., Heath L.S.: An $O(n\log n)$ algorithm for finding minimal perfect hash functions. Tech. Rep. 89-10, Dept. of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Va. 1989.

Recenzent: Dr Marek Tudruj

Wpłynęło do Redakcji 22 grudnia 1993 r.

Abstract

A parallel algorithm for conducting a search for a first solution to the problem of generating minimal perfect hash function is presented. The algorithm uses the technique of the processor farm. The basic concept of farming consists of having a central controller - the *Master* process - that hands out pieces of work to be processed by the members of a pool of *Worker* processes. The *Master* stores the active nodes of the search tree that represent the partial solution to the MPHF problem. The nodes are sent to the *Workers* which do the search by placing successive sets of words in the hash table and reporting the results to the *Master*.

The partial solutions are kept in a specially devised data structure that is called a *reversed trie* (*r-trie*). An r-trie is essentially a *b*-ary trie. The nodes on level *l* represent the set of partial solutions to the problem.

The parallel algorithm was implemented in occam on UKC's Meiko Surface. A farming harness was written specifically for the application (Fig. 3). The algorithm was run on a transputer system with ten T800 20 MHz transputers configured into a linear array. The *Master* was run on the first transputer, and the *Workers* were run on the remaining transputers. The experiments were conducted for six sets of words of sizes m = 50, 60,..., 100. For these sets Table 1 shows the execution time of the sequential search. Fig. 4 shows the speedup of the parallel algorithm as a function of the number of processors. The granularity of search work, s, is a parameter to the graphs. Each point of a graph was computed as an average over the 36 results measured for all the sets and the values of parameter β = 0.45, 0.46,..., 0.5.

The experiments showed that the parallel algorithm exhibits consistent and almost linear speedup. The r-trie structure proved to be very highly memory efficient.