ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1990
Seria: AUTOMATYKA z.100 Nr kol.1082

Leon Słomiński
Instytut Badań Systemowych PAN

IMPLEMENTACJE ALGORYTMÓW KOMBINATORYCZNYCH NA SYMULOWANYCH SIECIACH WIELOMIKROPROCESOROWYCH

Streszczenie. Przedstawiono wnioski z implementacji algorytmu mnożenia macierzy i algorytmu Prima-Dijkstry, znajdowania najkrótszych dróg i minimalnych dendrytów w digrafach (grafach) ważonych, na symulowanych strukturach wielomikroprocesorowych o architekturach: siatka, hipersześcian, potasowany stos, podwójne drzewo binarne i dolne drzewo binarne.

1. Wprowadzenie

Metody budowy i analizy algorytmów optymalizacji dla potrzeb obliczeń równoległych pozostają w tyle za rozwojem technologii i techniki budowy maszyn wieloprocesorowych. Wśród wielu powodów tego stanu wymienia się wyższy stopień trudności problematyki obliczeń równoległych, w porównaniu z obliczeniami w systemach sekwencyjnych. Wiele kłopotów sprawiają próby zdefiniowania jednolitego, akceptowanego przez teoretyków i praktyków, abstrakcyjnego modelu obliczeń równoległych. Jeden z powszechniej stosowanych w teorii złożoności obliczeń równoległych modeli - maszyna PRAM, nie uwzględnia kosztów komunikacji międzyprocesorowej. Wszystkie znane modele pomijają koszty operacji we/wy, co ma istotny wpływ na niedoszacowywanie nakładów czasowych obliczeń na maszynach równoległych. W tych warunkach dużą wagę przywiązuje się do eksperymentów laboratoryjnych i do stosowania symulacji obliczeń równoległych.

Uwagę naszą skupiamy na pewnej podklasie maszyn równoległych, której wyróżnikiem jest: - duża liczba (w założeniu, powyżej kilkuset) mikroprocesorów (μP), przy czym każdy z nich jest wyposażony w pamięć programu i danych i jest zdolny do wykonywania podstawowych operacji arytmetyczno-logicznych; - regularność, prostota (w tym stałość) połączeń między μP-rami, która gwarantuje bezpośrednią komunikację μP-ra ze stałą liczbą sąsiadów; - synchroniczny tryb pracy, polegający na jednoczesnym wykonywaniu, przez wszystkie μP-ry, tej samej instrukcji na różnych danych (model typu SIMD), przy czym synchronizm ten jest wymuszany przez procesor nadzorujący, który steruje strumieniem rozkazów i strumieniem danych.

W tej podklasie maszyn równoległych zajmujemy się następującymi strukturami: siatka wielowymiarowa, hipersześcian, potasowany stos, podwójne drzewo

*Praca powstała w ramach programu CPBP-02.15 koordynowanego przez IBS PAN.

binarne i dolne drzewo binarne. Wymienione struktury, wysoce reprezentatywne dla maszyn wielomikroprocesorowych (WμP), symulujemy na mikrokomputerze osobistym i wykorzystujemy do: - zaimplementowania algorytmu mnożenia macierzy, algorytmu Prima-Dijkstry dla znajdowania dendrytu minimalnego w grafie ważonym; - algorytmu Dijkstry - znajdowania drzewa dróg najkrótszych, z korzenia do wszystkich pozostałych wierzchołków, w digrafie ważonym; - przeanalizowania nakładów czasowych na komunikację wewnątrz struktury i na jej komunikację z otoczeniem oraz wpływu sposobu przedstawienia i rozmieszczenia danych o zadaniu na koszty czasowe realizacji wymienionych algorytmów. Na obecnym etapie badań przyjmujemy, że dysponujemy liczbą μP-rów, która pozwala realizować tzw. nieograniczony (co do zasobów) model obliczeń, co w praktyce oznacza, że rozmiar rozwiązywanych zadań dopasowujemy do liczby μP-rów, którą operuje symulator.

Wybrane przez nas struktury pozwalają zilustrować podstawowe mechanizmy uzyskiwania przyśpieszenia obliczeń: wektoryzację, konwejeryzację, powtarzalne połowienie, a także powtarzalne zdwajanie. Dobór zadań i algorytmów do ich rozwiązania ma na celu pokazanie, jak wymienione techniki wpływają na nakład obliczeń w przypadku występowania naturalnej równoległości (np. mnożenie macierzy) i w przypadkach, gdy algorytm jest niepodatny na zrównoleglanie (np. przegląd grafu w głąb lub wszerz - istota algorytmów Prima-Dijkstry i Dijkstry).

Zaznaczmy jeszcze, że pojęcie "struktura wielomikroprocesorowa" jest pojęciem dosyć szerokim i niejednoznacznym. Mikroprocesorem może być zarówno kilkurejestrowy element liczący, jak również transputer. Z toku prezentacji będzie wynikać, co rozumiemy przez pojęcie struktury WμP-wej.

Symulacji dokonujemy z pomocą symulatora Jupiter [3]. Jego możliwości teoretyczne określają parametry: do 512 μP-rów, do 16 kanałów połączeń na każdy μP-sor, do trzech odrębnych zadań na jeden μP-sor. Językiem programowania jest j. Parlan (pochodna j. Pascal).

W punkcie drugim pracy charakteryzujemy struktury wielomikroprocesorowe, które są przedmiotem naszego zainteresowania. Następnie (Punkt 3) opisujemy algorytmy i ogólne warunki ich implementacji na strukturach WμP-wych wykorzystujących wektoryzację, konwejeryzację i powtarzalne połowienie (zdwajanie). Ostatni punkt poświęcamy przedstawieniu ogólnych wniosków z implementacji wybranych algorytmów na pięciu sieciach wielomikroprocesorowych symulowanych na mikrokomputerze osobistym klasy XT/AT.

2. Charakterystyka wybranych struktur wielomikroprocesorowych

Idealizując, opisywane dalej struktury można sobie wyobrazić jako zbiory punktów w przestrzeni wielowymiarowej, połączonych między sobą w różny sposób, liniami, wzdłuż których punkty - mikroprocesory komunikują się między

sobą. Punkty oraz linie, które je łączą, tworzą sieć. Sieci różnią się między sobą sposobem połączeń między wierzchołkami. Ograniczamy się do sieci statycznych, to znaczy takich, które zachowują niezmienną topologię połączeń (w przeciwieństwie do sieci dynamicznych). Istotne znaczenie mają dwa parametry sieci: d_1 - stopień wierzchołka - maksymalna liczba bezpośrednich sąsiadów danego µP-ra; d_2 - średnica sieci - minimalna odległość (np. liczba linii) między najbardziej oddaloną między sobą parą µP-rów. Pożądane są sieci o d_1=const i o $d_2=O(\log p)$, gdzie p - liczba mikroprocesorów w sieci.

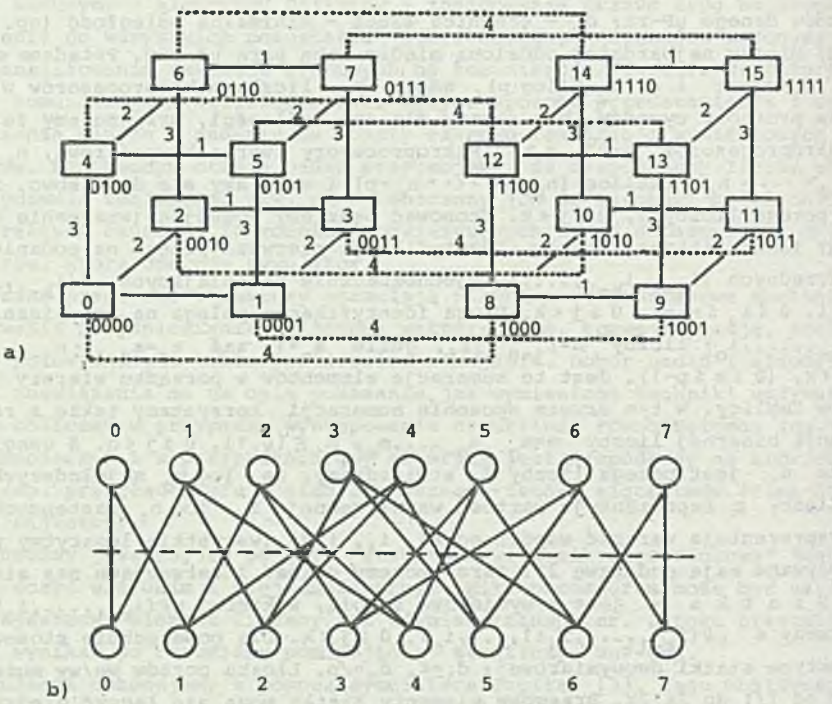
Dla prostoty wywodów, bez straty dla ich ogólności, przyjmujemy że liczba mikroprocesorów $p=2^q$, $q>1$. Mikroprocesory tworzą k-wymiarową, $n_{k-1} \times n_{k-2} \times \ldots \times n_o$, tablicę ($n_{k-1} \cdot \ldots \cdot n_o = p$) i umawiamy się dodatkowo, że n_j jest potęgą liczby 2, $0 \le j < k$. Stosować będziemy podwójną (wzajemnie równoważną) identyfikację µP-rów. Identyfikacja pierwsza polega na podaniu k współrzędnych $(i_{k-1}, i_{k-2}, \ldots, i_o)$ jednoznacznie określających $\mu P(i_{k-1}, i_{k-2}, \ldots, i_o)$, $0 \le i_j \le n_j - 1$, $0 \le j < k$. Druga identyfikacja polega na przypisaniu $\mu P(i_{k-1}, \ldots, i_o)$ liczby $m=\sum_{j=0}^{k-1} a_j i_j$, gdzie $a_o=1$, zaś $a_j = a_{j-1} \cdot n_{j-1}$, dla $1 \le j < k$, $(0 \le m \le p-1)$. Jest to numeracja elementów w porządku wierszy i wymiarów tablicy. W tym drugim sposobie numeracji korzystamy także z reprezentacji binarnej liczby $m=m_{q-1}m_{q-2}\ldots m_o$, $m_j \in \{0,1\}$, $0 \le j < q$. Z uwagi na to, że n_j jest potęgą liczby 2, stwierdzamy, że $\log n_o$ najmłodszych bitów liczby m reprezentuje wartość współrzędnej i_o, $\log n_1$ następnych bitów reprezentuje wartość współrzędnej i_1, itd. (wszystkie logarytmy przez nas używane mają podstawę 2). Teraz możemy opisać interesujące nas sieci.

S i a t k a - jest k-wymiarową siecią, w której $\mu P(i_{k-1}, \ldots, i_o)$ jest połączony z $\mu P(i_{k-1}, \ldots, i_j \pm 1, \ldots, i_o)$, $0 \le j < k$. Dla powszechnie stosowanej w praktyce siatki dwuwymiarowej: $d_1=4$, $d_2=\sqrt{p}$. Liczba portów we/wy może wynosić od 1/1 do 2k/2k. Brzegowe elementy siatki mogą się łączyć między sobą tworząc siatkę z domknięciem lub pozostać bez domknięcia.

H i p e r s z e ś c i a n - (rys.1a) jest k-wymiarową siecią, $k \ge 3$, w której µP-ry są umieszczone w węzłach k-wymiarowej kostki. µP(m) jest połączony bezpośrednio z q mikroprocesorami o numerach $m^{(b)}$, $0 \le b < q$, a $m^{(b)}=m_{q-1} \ldots m_{b+1} \overline{m}_b m_{b-1} \ldots m_o$ - powstaje z m przez zanegowanie bitu m_b. Dla hipersześcianu: $d_1=d_2=q=\log p$. Liczba portów we/wy wynosi od 1/1 do $O(\log p)/O(\log p)$.

P o t a s o w a n y s t o s (PS) jest to sieć, w której procesory łączą się ze sobą w sposób, który można przedstawić obrazowo jako idealnie potasowaną talię kart, złożoną z dwóch połówek. Wprowadza się następujące odwzorowania dla każdej liczby i: $SHUFFLE(i)=i_{q-2}i_{q-3}\ldots i_1 i_0 i_{q-1}$, (przesunięcie cykliczne o jeden bit w lewo w reprezentacji binarnej liczby i); $UNSHUFFLE(i)=i_o i_{q-1} i_{q-2} \ldots i_2 i_1$ (przesunięcie cykliczne o jeden bit w prawo w reprezentacji binarnej liczby i); $EXCHANGE(i)=i_{q-1}i_{q-2}\ldots\overline{i}_o$ (zanegowanie najmłodszego bitu). Mikroprocesor o numerze i jest połączony z mikroprocesorami o numerach: SHUFFLE(i), UNSHUFFLE(i), EXCHANGE(i). Sieć

PS można też przedstawić w postaci par numerów µP-rów, które mają bezpośrednie połączenie: (i,2i), (i+p/2,2i+1), (2i,2i+1) dla i=0,1,...(p/2)-1, (patrz Rys.1b). Dla sieci PS: $d_1=3$, $d_2=0(\log p)$, a liczba portów we/wy =1/1.



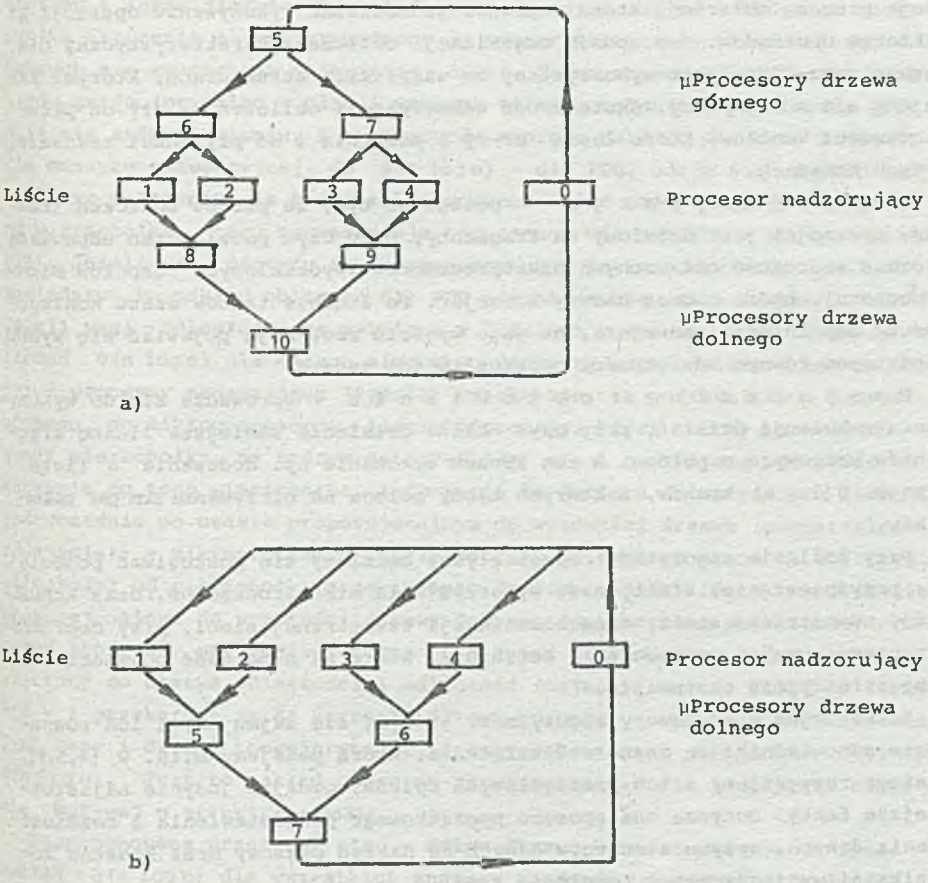
Rys.1.a). 4-wymiarowy hipersześcian, b). Potasowany stos. (Fig.1). a). 4-dimensional hipercube, b). Perfect Shuffle.

Uwaga: Sieć PS jest ośmiomikroprocesorowa. Drugą warstwę mikroprocesorów narysowano dla bardziej obrazowego przedstawienia istoty połączeń w potasowanym stosie (potasowanej talii kart).

P o d w ó j n e d r z e w o b i n a r n e (PDB) jest siecią hierarchiczną (patrz Rys.2a) utworzoną z dwóch drzew binarnych o wspólnych liściach. Drzewo górne tworzą µP-ry, których jedynym zadaniem jest przesłać informację otrzymaną na wejściu, do dwóch wyjść. Czas przejścia jednostki danych, z korzenia do liści, jest proporcjonalny do wysokości drzewa $h=\log n$, gdzie n jest liczbą liści (będziemy przyjmować, że n jest potęgą liczby 2). Liście reprezentują mikroprocesory zdolne do wykonywania podstawowych

działań arytmetyczno-logicznych oraz prostych programów. Drzewo dolne tworzą µP-ry, które wykonują proste operacje dwuoperandowe, typu +, -, ×, ÷, max, min itp. na operandach otrzymanych na wejściach, a wyniki przekazują do wyjścia. Procesor nadzorujący steruje pracą sieci. Łączna liczba µP-rów w PDB wynosi $p=3\cdot n-1$ (uwzględniając procesor nadzorujący), $d_1=3$, $d_2=0(\log p)$. Liczba portów we/wy wynosi 1/1.

Rysunek 2b pokazuje sieć nazwaną d o l n e d r z e w o . b i n a r n e, o której mowa będzie w następnym punkcie.



Rys. 2. a), Podwójne drzewo binarne, b), Dolne drzewo binarne. (Fig.2), a), Double-binary tree, b), Lower binary tree.

3. Algorytmy i ogólne warunki ich implementacji na sieciach wielomikroprocesorowych

Przyśpieszenie obliczeń na maszynach wieloprocesorowych uzyskuje się wieloma sposobami. W naszych rozważaniach mają zastosowanie trzy z nich: wektoryzacja, konwejeryzacja, powtarzalne połowienie (jego odwróceniem jest powtarzalne zdwajanie). Każdy z tych sposobów może być zastosowany na poziomie mikrorozkazów, instrukcji lub programu. Nasze zainteresowania dotyczą dwóch ostatnich poziomów. W e k t o r y z a c j a to taka organizacja procesu obliczeń, która zapewnia jednoczesne wykonywanie operacji na wektorze operandów. Ten sposób organizacji obliczeń, charakterystyczny dla maszyn typu SIMD, jest wykorzystany we wszystkich strukturach, którymi zajmujemy się w tej pracy. Skuteczność wektoryzacji obliczeń zależy od przepustowości kanałów, które łączą μP-ry z pamięcią i od płynności transmisji w tych kanałach.

K o n w e j e r y z a c j a - polega na tym, że proces obliczeń (rozkaz, operacja) jest dzielony na fragmenty, przy czym podział ten odpowiada liczbie szeregowo połączonych mikroprocesorów (wydzielonych zespołów mikroprocesora), które tworzą umowny konwejer. Po liczbie taktów czasu koniecznych do zapełnienie konwejera, na jego wyjściu zaczynają pojawiać się wyniki z odstępem równym odwrotności częstotliwości zegara.

P o w t a r z a l n e p o ł o w i e n i e - sprowadza się do wykonywania sekwencji działań, przy czym każde działanie zmniejsza liczbę argumentów końcowych o połowę. W ten sposób wykonanie np. dodawania n liczb wymaga $O(\log n)$ kroków, z których każdy polega na otrzymaniu sum par składników.

Przy analizie algorytmów równoległych, będziemy się posługiwać pojęciami: przyśpieszenie, efektywność wykorzystania mikroprocesorów, czas komunikacji wewnętrznej sieci, czas komunikacji zewnętrznej sieci, przy czym zrezygnujemy tutaj z przytoczenia definicji, które są albo dość powszechne albo intuicyjnie zrozumiałe.

Sekwencyjne pierwowzory algorytmów, którymi się zajmujemy i ich równoległe odpowiedniki są znane w literaturze, którą podajemy m.in. w [4,5,6], dlatego rezygnujemy z ich szczegółowych opisów, podając jedynie najistotniejsze fakty. Dotyczą one sposobu początkowego przedstawienia i rozmieszczenia danych, wpływu struktury danych na nakład czasowy oraz kosztów komunikacji wynikających z topologii sieci.

Do mnożenia korzystamy z naturalnego algorytmu wynikającego z definicji iloczynu macierzy: $C = A \times B$, gdzie $c_{ij} = \sum_{l=1}^{n} a_{il} b_{lj}$, a n jest wymiarem macierzy. Dla znalezienia dendrytu o najmniejszej wadze w grafie ważonym korzystamy z algorytmu Prima- Dijkstry. Z algorytmu Dijkstry korzystamy do znalezienia drzewa dróg najkrótszych, z wybranego wierzchołka do wszy-

stkich powstałych wierzchołków, w ważonym grafie skierowanym. Zakładamy, że grafy i digrafy, z którymi mamy do czynienia, spełniają warunki dopuszczające stosowanie wymienionych algorytmów. Rozmiar grafu będziemy przedstawiać liczbą jego wierzchołków n. Równoległe algorytmy Prima-Dijkstry i Dijkstry mają identyczną strukturę, ich implementacje na sieciach drzewiastych nastręczają identyczne problemy prowadząc do tych samych wniosków. Dlatego ograniczymy się dyskusji jedynie drugiego z nich.

Istota algorytmu Dijkstry polega na uporządkowanym przeglądzie wierzchołków i łuków digrafu. Przegląd ten rozpoczyna się od ustalonego wierzchołka - korzenia i jest prowadzony do wyczerpania listy wierzchołków, do których nie wyznaczono najkrótszej drogi. Znane sekwencyjne sposoby przeglądu grafu (przegląd w głąb i wszerz) tylko w ograniczonym stopniu poddają się zrównolegleniu. Z tego powodu poprawę nakładu obliczeń z $O(n^2)$ - dla maszyny sekwencyjnej, do $O(n \log n)$ - dla PDB, udaje się uzyskać jedynie przy założeniu, że każdy mikroprocesor - liść może obliczyć odległość od wierzchołka, który reprezentuje, do innego wierzchołka w stałym czasie $O(1)$. Jeżeli wagi digrafu są reprezentowane w postaci macierzy lub list sąsiedztwa, to nakład obliczeń dla PDB pozostaje bez zmian i wynosi $O(n^2)$. Jeżeli wagi - odległości są ustalane w liściach w czasie $O(1)$, to nakład obliczeń $O(n \log n)$ dla całego algorytmu Dijkstry uzyskuje się w ten sposób, że: - procesor nadzorujący (Rys.2a) przekazuje, za pośrednictwem drzewa górnego, do mikroprocesorów - liści informację, którą tworzy para liczb - numer wierzchołka, na którym zatrzymaliśmy się i najkrótsza odległość od korzenia do tego wierzchołka. Informacja ta dociera do wszystkich liści jednocześnie po czasie proporcjonalnym do wysokości drzewa (powtarzalne zdwajanie); - mikroprocesory-liście potrzebują stałego czasu na obliczenie odległości od wierzchołka przekazanego do wierzchołka-liścia; - po czasie proporcjonalnym do wysokości drzewa dolnego (powtarzalne połowienie) procesor sterujący otrzymuje informację - numer wierzchołka, który ma być włączony do drzewa odległości i odległość (najmniejszą) do niego od korzenia - i przekazuje ją do korzenia drzewa górnego; - cykl ten powtarza się $O(n)$ razy i dzięki konwejeryzacji, łączny czas zajęty przez cykle jest $O(n \log n)$. Jest to nakład czasu na rozwiązanie naszego zadania na maszynie WμP-wej o strukturze PDB.

Zaproponowana przez nas sieć - dolne drzewo binarne pozwala otrzymać nakład $O(n \log n)$ dla wszystkich znanych sposobów przedstawiania danych o digrafie (grafie). Jedna instrukcja wektorowa powoduje przekazanie wektora danych o długości n z pamięci procesora sterującego do wszystkich (lub wybranych) liści. Dalszy przebieg algorytmu jest bez zmian.

Mnożenie macierzy poddaje się łatwo zrównoleglaniu na maszynach o różnej konfiguracji i liczbie mikroprocesorów. Rozważmy dwa sposoby mnożenia macierzy na siatce dwuwymiarowej (k=2) o n^2 μP-rach.

a) Mnożenie systoliczne. Zakłada się, że dwa strumienie danych zasilają sieć (przyjmujemy siatkę bez domknięcia). Strumień pierwszy tworzą wiersze macierzy A, przesunięte w czasie względem pierwszego wiersza o jeden, dwa itd. elementów. Tak uszeregowane dane są przekazywane na "wierszowe" wejścia siatki. Drugi strumień tworzą kolumny macierzy B, podobnie uszeregowane i podawane na wejścia "kolumnowe" siatki. W pierwszym takcie do $\mu P(1,1)$ dociera jednocześnie para (a_{11},b_{11}); mikroprocesor wykonuje mnożenie: $a_{11}b_{11}$, dodając wynik do zawartości akumulatora (przed rozpoczęciem pracy wszystkie akumulatory zostają wyzerowane). W drugim takcie obliczenia wykonują mikroprocesory: $\mu P(1,1)$ - na danych (a_{12},b_{21}), $\mu P(1,2)$ - na danych (a_{11},b_{12}) i $\mu P(2,1)$ - na danych (a_{21},b_{11}). Wynikiem drugiego kroku są sumy cząstkowe dla elementów wynikowych w akumulatorach trzech μP-rów. Nietrudno stwierdzić, że po $(2n-1)$ krokach wynik mnożenia jest zapisany w akumulatorach mikroprocesorów siatki, tzn., że: $c_{ij}=\Sigma_{l=1}^{n}a_{il}b_{lj}=\mu P(i,j)$, (utożsamiamy symbol μP-ra z jego akumulatorem). Łączny nakład obliczeń składa się z $O(n)$ operacji przesłań i $O(n)$ operacji dodawania. Nakład obliczeń algorytmu równoległego mnożenia dwóch macierzy na niedomkniętej siatce o n^2 μP-rach wynosi $O(n)$, przyśpieszenie - $O(n^2)$, efektywność wykorzystania μP-rów - $O(1)$.

Zauważmy, że wyprowadzenie macierzy C na zewnątrz sieci wymaga dodatkowo $O(n)$ operacji, przy wykorzystaniu n portów wyjściowych.

b) Mnożenie rezydentalne. Zakłada się, że przed rozpoczęciem algorytmu podstawowego elementy macierzy A i B są odpowiednio rozmieszczone w rejestrach μP-rów siatki (hipersześcianu, PS). Wprowadzenie danych do sieci może kosztować od $O(n)$ do $O(n^2)$ operacji, zależnie od liczby wejść sieci. Pozostaniemy przy siatce dwuwymiarowej $(p=n^2)$, z domknięciem mikroprocesorów brzegowych ($\mu P(1,1)$ jest połączony dodatkowo z $\mu P(1,n)$ i $\mu P(n,1)$, $\mu P(1,2)$ jest połączony z - $\mu P(1,n)$, itd.). Każdy mikroprocesor ma trzy rejestry, które będziemy oznaczać literami A(i,j), B(i,j), C(i,j). Implementujemy dwufazowy algorytm zaproponowany przez Cannona, [1]. Faza pierwsza - zawartość rejestru $A(i,j)=a_{ij}$ podlega cyklicznemu przesunięciu w lewo o i pozycji, i=1,...,n. Podobnie zawartość rejestru $B(i,j)=b_{ij}$ podlega przesunięciu cyklicznemu o j pozycji w górę, j=1,...,n. Faza druga - (n-1) razy wykonywane są następujące działania: - zawartość każdego rejestru A(i,j) przesuwa się o jedno miejsce w lewo; - zawartość każdego rejestru B(i,j) przesuwa się o jedno miejsce w górę; - po każdym przesunięciu wykonać operacje: $A(i,j)\cdot B(i,j)$, $C(i,j):=C(i,j)+A(i,j)\cdot B(i,j)$. W obydwóch fazach operacje są wykonywane jednocześnie dla i=1,...,n, a następnie jednocześnie, dla j=1,...,n. Po wykonaniu wszystkich działań mamy: $C(i,j)=c_{ij}=\Sigma_{l=1}^{n}a_{il}\cdot b_{lj}$. Nakład obliczeń na komunikację wewnętrzną jest równy $4(n-1)$; każdemu przekazaniu zawartości z rejestru do rejestru, w drugiej fazie, towarzyszą dwie operacje arytmetyczne. Oczywista, nakład czasu dla całego algorytmu jest

0(n) - podobnie jak dla mnożenia systolicznego.

Mnożenie rezydentalne na hipersześcianie i na PS o n^2 lub n^3 mikroprocesorach przebiega również w dwóch fazach: wewnętrzne przesunięcia między rejestrami, operacje pomnóż-dodaj z towarzyszącym przesunięciem. Rozmieszczenie początkowe i przesunięcia między rejestrami są odmienne w każdym z przypadków [1, 2, 7].

W tablicy 1 zestawiliśmy najważniejsze parametry charakteryzujące sieć i odpowiedni algorytm (mnożenia macierzy i algorytm Dijkstry).

Tablica 1

Sieć / Algorytm	Liczba mikroprocesorów p	Koszt komunikacji	Koszt pracy µP-rów	Koszt algorytmu	Przyśpieszenie / Wykorzyst. µP-rów
Siatka (bez domknięcia) Mnożenie systoliczne	n^2	2n-1	0(n)	0(n)	$0(n^2)$ 0(1)
* Siatka (z domknięciem) Mnożenie rezydentalne	n^2	4(n-1)	0(n)	0(n)	$0(n^2)$ 0(1)
* Hipersześcian Mnożenie rezydentalne	n^2	2(q+n-1)	0(n)	0(n)	$0(n^2)$ 0(1)
Hipersześcian Mnożenie rezydentalne	n^3	5 log n	0(log n)	0(log n)	$0(n^3/\log n)$ $0(1/\log n)$
PS. Mnożenie rezydentalne	n^2	0(n)	0(n)	0(n)	$0(n^2)$ 0(1)
* PS. Mnożenie rezydentalne	n^3	10 log n	0(log n)	0(log n)	$0(n^3/\log n)$ $0(1/\log n)$
* PDB. Algorytm Dijkstry	3n-1	~2h	0(n)	0(n log n)	0(n/log n) 0(1/log n) lub żadne
* DDB. Algorytm Dijkstry	2n	~1h	0(n)	0(n log n)	0(n/log n) 0(1/log n)

* Oznaczono algorytmy, które były implementowane przez nas.

4. Implementacja symulacyjna na mikrokomputerze

Korzystając z symulatora sieci wielomikroprocesorowych zainstalowanego na mikrokomputerze osobistym PC/AT, zaimplementowano algorytmy wymienione

w punkcie 3. W tablicy 2 przedstawiono dane dotyczące algorytmu, typu sieci, rozmiarów zadań i liczby mikroprocesorów tworzących sieć.

Tablica 2

Zadanie	Algorytm	Sieć	Rozmiar zadań, n	Liczba μP-rów p
Mnożenie macierzy	$c_{ij}=\sum_{l=1}^{n} a_{il} b_{lj}$	Siatka n^2	4,8	16,64
		Sześcian n^2	4,8	16,64
		PS n^3	4	64
Dendryt minimalny	Prima-Dijkstry	Podwójne drzewo binarne	4, 8, 16	(p=3n-1)=11,23,47
Najkrótsze drogi	Dijkstry	Podwójne drzewo binarne	4, 8, 16	〃
		Dolne drzewo binarne	4, 8	(p=2n)=8, 16

Zadania testowe były generowane losowo: w przypadku PDB i DDB - w mikroprocesorze nadzorującym; dla pozostałych sieci - przez mikroprocesory sieci (siatka, hipersześcian) lub poza siecią (PS), z umieszczeniem danych w sieci przed rozpoczęciem algorytmu. W tablicach 3 i 4 zebrano dane charakteryzujące pracę mikroprocesorów: czas pracy, efektywność - mierzona stosunkiem czasu pracy rzeczywistej (bez czasu oczekiwania) do całego czasu, liczba instrukcji. W każdym przypadku generowano po 3-4 zadania o różnych wartościach n.

Tablica 3

n	Czas pracy w $[ms]\cdot 10^{-3}$.Od-do		Efektywność	Efektywność	Liczba instrukcji
	μP-rów	μP-ra nadzor.	μP-rów	μP-ra nadzor	od - do
Najkrótsze drogi, sieć - PDB					
4	32 - 35	37 - 38	0,13 - 0,25	0,73	180 - 200, 980
8	124 - 130	135 - 136	0,07 - 0,29	0,75	600 - 1380, 3700
16	475 - 488	497 - 498	0,04 - 0,32	0,80	1000 - 5600, $1,4\cdot 10^4$
Najkrótsze drogi, sieć - DDB					
4	40 - 42	45	0,10 - 0,25	0,74	160 - 350, 1200
8	141 - 145	150	0,06 - 0,16	0,78	300 - 700, 4400
Dendryt minimalny, sieć - PDB					
4	27 - 32	34 - 35	0,11 - 0,23	0,70	150 - 190, 900
8	100 - 109	113 - 114	0,07 - 0,27	0,71	-
16	355 - 367	377	0,04 - 0,32	0,77	-

Tablica 4

Mnożenie macierzy									
n	Czas pracy µP-ra [ms]·10^{-3}			Efektywność µP-rów			Liczba instrukcji / µP-sor		
	SIATKA	HIPERSZEŚC.	PS	SIATKA	HIPERSZEŚC.	PS	SIATKA	HIPERSZEŚC.	PS
4	11-12	18-19	19-21	0,97-1,0	0,93-1,0	0,6-1,0	381-450	666-710	415-780
8	19-22	31	-	0,86-1,0	0,90-1,0	-	623-800	1040-1120	-

Przytaczając, w tablicach 3 i 4, wyniki implementacji zwracamy uwagę na ich ograniczony zakres, co nakazuje zachować ostrożność w wyciąganiu wniosków o charakterze ilościowym. Niepełne wykorzystanie teoretycznych możliwości symulatora, wynika z faktu, że dla większej liczby mikroprocesorów jego zachowanie bywa nieokreślone, co ma swoje źródło w niezidentyfikowanych błędach. Przyjęty model obliczeń z nieograniczonymi zasobami nie daje możliwości rozwiązywania zadań ze zbyt dużą wartością n.

Zwraca uwagę niska i malejąca ze wzrostem n efektywność wykorzystania µP-rów w sieciach PDB i DDB. Jest to wynik spodziewany (tabl.1), a konkretne wartości zależą od digrafu. Mikroprocesory, które wykonały swoje zadanie, pauzują w dalszych iteracjach, co pogarsza ich efektywność. W rubryce "Liczba instrukcji", wartość podana po przecinku (tab.3). odnosi się do procesora nadzorującego, który jest stosunkowo wysoko obciążony. Szczegółowa analiza diagramów [5, 6] pokazuje, że w sieci DDB mikroprocesory są wykorzystywane znacznie bardziej równomiernie, niż to ma miejsce w sieci PDB.

Analiza zachowania się sieci SIATKA i HIPERSZEŚCIAN w odniesieniu do mnożenia macierzy jest bardzo podobna. Warunki wykorzystania mikroprocesorów sprzyjają ich efektywności. Mniejszą efektywność i większą wrażliwość na rozdział zadań między µP-ry wykazuje sieć POTASOWANY STOS.

Za największy niedostatek symulatora należy uznać brak możliwości symulacji różnych metod, trybów i protokołów komunikacji oraz brak możliwości pomiaru czasu komunikacji. Czas oczekiwania mikroprocesorów, mający wpływ na efektywność ich wykorzystania, zawiera w sobie zarówno czas komunikacji, jak i czas przestoju związany z algorytmem, zadaniem i z realizacją algorytmu. Problemy te wymagają odrębnego przestudiowania.

5. Podsumowanie

Przedstawiono wyniki (Tablice 2-4) ograniczonego testowania wybranych algorytmów optymalizacji dyskretnej na symulowanych strukturach wielomikroprocesorowych o architekturach: siatka, hipersześcian, potasowany stos, drzewo binarne. Jednym z ważniejszych celów eksperymentu była obserwacja nierównomierności obciążenia µP-rów i efektywności ich wykorzystania (Tablice 3 i 4). Badanie algorytmów i struktur wieloprocesorowych dla ich realizacji połączono z:

- oceną stopnia wykorzystania zasadniczych technik algorytmicznych zrównoleglania obliczeń - konwejeryzacji, wektoryzacji i powtarzalnego połowienia w wymienionych strukturach;
- analizą wpływu użytych struktur danych,sposobu rozmieszczenia danych początkowych, trybu pracy (systoliczny lub rezydentny) struktury oraz liczby portów we/wy,na stałe wchodzące do asymptotycznych szacunków nakładów obliczeń według oceny najgorszego przypadku;
- oceną nakładów na komunikację międzyprocesorową i ich udziału w ogólnych nakładach obliczeń.

Wyniki tych rozważań można ująć następująco:

- Najbardziej uniwersalną techniką algorytmiczną zrównoleglania obliczeń jest konwejeryzacja (korzysta się z niej intensywnie we wszystkich strukturach). Konwejeryzacja umożliwia wykonanie N operacji na S -segmentowym konwejerze, przy użyciu T jednostek czasu na pojedynczą operację, w łącznym czasie (N+S-1)T, co daje O(S)- krotne przyśpieszenie w stosunku do czasu obliczeń w układzie sekwencyjnym.
- Wektoryzacja jest wykorzystywana w pełni w strukturze siatki, hiperześcianu i struktury PS, przy założeniu że algorytm dopuszcza operacje na wektorze danych (ma to miejsce np. w przypadku mnożenia macierzy), oraz jej wykorzystanie jest ograniczone (zasadniczo do μP-rów liści) w strukturach drzewiastych (mniej w PDB, więcej w DDB). Zastąpienie podwójnego drzewa binarnego, strukturą dolnego drzewa binarnego, z wektorowym wprowadzaniem danych z procesora sterującego do liści, zmniejsza o połowę czas komunikacji międzyprocesorowej (Tablica 3).
- Powtarzalne połowienie (zdwajanie)jest efektywnie wykorzystywane wówczas, gdy na N - elementowym zbiorze danych mamy wykonać ciąg operacji dwuargumentowych, których wynikiem ma być jedna wartość (np. dodawanie N elementów, znalezienie elementu o wartości największej lub najmniejszej w zbiorze itp). Powtarzalne połowienie ma nakład obliczeń O(log N) i jest obok konwejeryzacji podstawowym sposobem zrównoleglania działań w strukturach drzewa binarnego.
- Liczba portów we/wy ma zasadniczy wpływ na czas wprowadzania danych i wyprowadzania wyników w strukturach wieloprocesorowych. Jeden biegun stanowią wieloportowe struktury siatki μP-rów, drugi biegun to struktura PDB i PS. Dolne drzewo binarne ma n portów wejścia i jedno wyjście (komunikacja z procesorem sterującym). Struktury wieloportowe umożliwiają systoliczny tryb pracy, który stwarza najkorzystniejszy bilans nakładów obliczeń w relacji czas pracy μP-rów - czas komunikacji międzyprocesorowej. Zauważmy, że systoliczne mnożenie macierzy na siatce zapewnia, że każdemu taktowi przesłania danych towarzyszy takt wykonania operacji. W strukturze hiperześcianu, PS i drzew binarnych jednej operacji obliczeń towarzyszy wiele operacji przesłań, co zwiększa koszty komunikacji (Tablica 3).

- Zaproponowana struktura DDB umożliwia zachowanie ogólnego nakładu obliczeń O(nlog n) dla algorytmu Dijkstry znajdowania dróg najkrótszych z wybranego wierzchołka do wszystkich pozostałych wierzchołków grafu i dla algorytmu Prima-Dijkstry znajdowania dendrytu minimalnego, przy powszechnie stosowanych strukturach danych przedstawiających wagi łuków (macierz, listy sąsiedztwa). Drzewa binarne są przykładem struktur dedykowanych dla realizacji algorytmów opartych na przeglądzie grafu - niepodatnych na zrównoleglanie. Tego typu algorytmy implementowane na strukturach ogólnego przeznaczenia (siatka, hipersześcian) mają większą efektywność w warunkach niesynchronicznej pracy struktury. Znajdowanie dróg najkrótszych między wszystkimi parami wierzchołków grafu można zrealizować efektywnie na strukturach wieloprocesorowych przez wykorzystanie algorytmu mnożenia macierzy i jego podatności na zrównoleglenie.

Przedstawiliśmy fragment wieloaspektowych i wielopłaszczyznowych badań dotyczących algorytmów optymalizacji dyskretnej w warunkach systemów obliczeń równoległych. Wprowadzającym przewodnikiem po literaturze przedmiotu może być pozycja [8] , będąca komentowaną bibliografią prac poświęconych optymalizacji równoległej.

LITERATURA

[1] Dekel E., Nassimi D., Sahni S.: Parallel matrix and graph algorithms SIAM J. on Compt. v. 10, No.4, 1981, pp. 653-675.

[2] Gondek H.: Równoległe algorytmy mnożenia macierzy. Implementacja i eksperyment obliczeniowy na symulowanych strukturach typu MESH i HYPERCUBE o n^2 procentach. ZPM-11/A1530/89. Raport IBS PAN Warszawa.

[3] JUPITER. Podręcznik użytkownika. Abaks PTH, Kraków 1987.

[4] Kaliszewski I., Kulczycka D., Słomiński L.: Równoległy algorytm Prima-Dijkstry wyznaczania dendrytu minimalnego: Implementacja na strukturze typu dwudrzewo. ZPM-5/A1530/88. Raport IBS PAN, Warszawa.

[5] Słomiński L., Gondek H.: Implementacja na symulowanej maszynie równoległej typu dwudrzewo algorytmu Dijkstry wyznaczania drzewa dróg najkrótszych w digrafie.ZPM-37/A1530/88. Raport IBS PAN, Warszawa.

[6] Słomiński L., Gondek H.: Symulowane odmiany struktury dwudrzewa dla algorytmu dróg najkrótszych w digrafie. ZPM-10/A1530/89. Raport IBS PAN, Warszawa.

[7] Sobczyńska J.: Charakteryzacja wieloprocesorowej struktury PERFECT SHUFFLE. Równoległy algorytm mnożenia macierzy i jego implementacja na tej strukturze. ZPM-9/A1530/89. Raport IBS PAN, Warszawa.

[8] Zenios S.A.: An annotated bibliography on parallel optimization. DSD, University of Pennsylvania, Philadelphia, PA 19104. Report 87-12-04. December 1987.

Recenzent: Prof.dr hinż.J.Błażewicz

Wpłynęło do Redakcji do 1990-04-30.

IMPLEMENTATIONS OF COMBINATORIAL ALGORITHMS ON SIMULATED MULTIMICROPROCESSOR NETWORKS

S u m m a r y

Conclusions derived from implementation of an algorithm of matrix multiplication, and Prim-Dijkstra's algorithm of finding one-to-all shortest paths tree and minimum spanning tree in weighted digraphs (graphs), by means of simulated multimicroprocessor networks like: mesh, hypercube, perfect shuffle, double binary tree and lower binary tree, are presented.

РЕАЛИЗАЦИЯ КОМБИНАТОРНЫХ АЛГОРИТМОВ НА СИМУЛИРОВАННЫХ МНОГОМИКРОПРОЦЕССОРНЫХ СЕТЯХ

Р е з ю м е

Представлены выводы реализации алгоритма умножения матриц и алгоритма Прима-Дийскры для нахождения кратчайших путей и минимальных дендритов во взвешенных диграфах /графах/ на симулированных многомикропроцессорных структурах с архитектурой: решётка, гиперкуб калиевый штабель, двойное бинарное дерево и нижнее бинарное дерево.