Problemy równoległej optymalizacji dyskretnej

PROBLEMY RÓWNOLEGŁEJ OPTYMALIZACJI DYSKRETNEJ

Redaktorzy:

Leon Słomiński

Ignacy Kaliszewski

1. Wprowadzenie

Niniejsza książka prezentuje wyniki prac prowadzonych w Zakładzie Programowania Matematycznego Instytutu Badań Systemowych PAN w latach 1986-92, w ramach tematu badawczego "*Optymalizacja na strukturach kombinatorycznych z uwzględnieniem obliczeń równoległych*". Podjęcie tego tematu, którym kierował pierwszy z współredaktorów, poprzedziło roczne wspólne seminarium Zakładu Programowania Matematycznego IBS PAN oraz Pracowni Metod Numerycznych Instytutu Podstaw Informatyki PAN. Niektóre pomysły dyskutowane podczas seminarium znalazły swoje odbicie w treści tej książki.

Celem książki jest przybliżenie polskiemu Czytelnikowi problematyki obliczeń równoległych w zadaniach optymalizacji, a specjalnie w zadaniach optymalizacji dyskretnej oraz uzupełnienie, chociażby częściowe, luki jaka jest w tym zakresie zauważalna w krajowej literaturze. Z publikacji w języku polskim poświęconych w całości lub w części metodom i algorytmom równoległym optymalizacji dyskretnej można wymienić pozycje [1 ÷ 3]. Postawiony cel chcemy osiągnąć przez zaprezentowanie wyboru zadań optymalizacyjnych z zakresu naszych zainteresowań i przedstawienie, na ich przykładzie, problemów związanych z konstrukcją algorytmów równoległych. Problemy te ukazujemy na tle znacznie szerszego wachlarza zagadnień związanych z obliczeniami równoległymi.

Książka ma następujący układ. Rozdział drugi zawiera podstawowe pojęcia związane z obliczeniami równoległymi i z maszynami równoległymi. Przedstawiono w nim najważniejsze modele obliczeń równoległych, typowe architektury maszyn równoległych i ich charakterystyki, a także podstawowe wyniki teorii złożoności dla

algorytmów równoległych. Dopełnieniem tego rozdziału jest omówienie nowatorskiej realizacji sprzętowej - transputera, którego sieci wydają się szczególnie przydatne do rozwiązywania zadań optymalizacji dyskretnej za pomocą ogólnych i wąsko ukierunkowanych algorytmów równoległych. W rozdziale trzecim przedstawiono algorytmy równoległe dla rozwiązywania zadania wyznaczania dendrytu minimaksowego w grafie skierowanym z wagami na łukach. W celu pokazania związku, który zachodzi między złożonością algorytmu i wybranym modelem maszyny równoległej, użyto różnych modeli maszyny i różnych pierwowzorów algorytmów sekwencyjnych. Rozdział czwarty przedstawia algorytmy wyznaczania elementu maksymalnego w skończonym zbiorze wektorów oraz opis ich realizacji na sieci transputerów, wraz z wynikami testów numerycznych. W rozdziale piątym opisano asynchroniczny algorytm równoległy, do rozwiązywania algebraicznego zagadnienia k - przydziału, przedstawiono realizację tego algorytmu na sieci transputerów oraz wyniki eksperymentu numerycznego. Kolejny, szósty rozdział prezentuje język programowania równoległego MODEST. Jest to język ogólnego zastosowania zawierający mechanizmy do uruchamiania procesów i kontroli przebiegu obliczeń równoległych. Dwa dodatki zawierają: Dodatek 1 - Polsko - angielski słownik nazw i pojęć z zakresu obliczeń równoległych, Dodatek 2 - Listę artykułów i raportów ZPM IBS PAN, które ukazały się w latach 1986 - 1992 i które dotyczą problematyki tej książki.

Warszawa, styczeń 1993.

Leon Słomiński
Ignacy Kaliszewski

Literatura

1. Błażewicz J. (1988): Złożoność obliczeniowa problemów kombinatorycznych. WNT, Warszawa.
2. Słomiński L., Kaliszewski I. (1988): "Problemy obliczeń równoległych". *Prace IBS PAN*, 168, Warszawa.
3. Sysło M.M. (1985): "Maszyny i algorytmy równoległe". *Raport Instytutu Informatyki Uniwersytetu Wrocławskiego*, 154, Wrocław.

4. Wyznaczanie elementów maksymalnych w zbiorach skończonych z wykorzystaniem sieci transputerów

4.1. Wstęp

W rozdziale tym zajmiemy się wyznaczaniem elementów maksymalnych w zbiorach zawierających skończoną ilość wektorów (elementów przestrzeni R^k). Znane z literatury algorytmy sekwencyjne ([7],[8]) są wystarczająco efektywne dla szerokiej klasy zadań. Przykładowo, algorytm podany w pracy [7] wymaga w najgorszym przypadku $0.5\ k\ |N|\ (\ 2\ |Z| - k - 1\ )$ porównań, gdzie $|Z|$ oznacza liczbę elementów zbioru Z , $|N|$ liczbę elementów maksymalnych w tym zbiorze, k długość wektorów reprezentujących elementy zbioru (liczbę współrzędnych). Istnieją jednak problemy, w których wyznaczanie maksymalnych elementów jest kluczowym ogniwem obliczeń, stąd też tendencja do poszukiwania algorytmów jeszcze bardziej efektywnych. Przykładami takich problemów mogą być zadania wielokryterialnego programowania dynamicznego [9], wielokryterialnego programowania całkowitoliczbowego [1], wielokryterialnej optymalizacji globalnej [2].

W pracy [4] przedstawiono równoległy algorytm dla wyznaczania wszystkich maksymalnych elementów w zbiorach skończonych. Algorytm ten został przetestowany za pomocą pakietu symulacyjnego ([4]), a następnie na strukturze wieloprocesorowej (transputery) ([5]). Algorytm ten używa k wzajemnie komunikujących się procesów (*Communicating Sequential Processes* [3]). Postulowaną architekturą maszyny równoległej dla tego algorytmu jest topologia pełnego grafu. Z faktu, że średnica pełnego grafu wynosi 1, wynika że prędkość przesyłania informacji pomiędzy dowolną parą takich

procesorów jest ograniczona jedynie przez właściwości sprzętowe. Głównym celem tego rozdziału jest przedstawienie rezultatów testowania algorytmu na rzeczywistej maszynie równoległej.

4.2. Sformułowanie problemu

Przez R^k będziemy oznaczać przestrzeń rzeczywistą z porządkiem częściowym generowanym przez stożek R_+^k, $R_+^k = \{y \in R^k \mid y_i \geq 0,\ i = 1, \ldots, k\}$, w następujący sposób: elementy $y, y' \in R^k$ są w relacji porządku, tj. $y \leq y'$, wtedy i tylko wtedy, gdy $y' - y \in R_+^k$.

Jeżeli $y, y' \in R^k$, $y' \neq y$, oraz $y \leq y'$, to mówimy, że w sensie relacji element y jest mniejszy od y' a element y' jest większy od y.

Niech Z będzie pewnym podzbiorem R^k. Mówimy, że element y zbioru Z jest *maksymalny*, jeżeli nie istnieje element y', $y' \in Z$, $y' \neq y$, taki, że $y \leq y'$.

Ponieważ porządek częściowy jest generowany przez stożek R_+^k, powyższa definicja elementów maksymalnych jest równoważna następującej definicji. Element $y \in Z$ nazywamy maksymalnym, jeżeli nie istnieje element $y' \in Z$, $y' \neq y$, taki, że $y_i \leq y_i'$, $i = 1, \ldots, k$.

W dalszym ciągu tego rozdziału będziemy zakładać, że Z jest zbiorem skończonym.

4.3. Algorytm sekwencyjny

Przedstawimy teraz algorytm sekwencyjny dla wyznaczania elementów maksymalnych w zbiorze skończonym. Spośród algorytmów znanych z literatury algorytm ten ma najmniejszą złożoność. Algorytm ten stanowił punkt wyjścia dla konstrukcji algorytmu równoległego, który przedstawimy w następnym paragrafie.

Podstawową ideą wykorzystywaną przy wyznaczaniu elementów maksymalnych w zbiorze skończonym jest fakt, że wśród elementów zbioru, których dowolna, ustalona współrzędna przyjmuje wartość maksymalną, znajduje się element maksymalny zbioru.

Wszystkie elementy maksymalne zbioru Z można wyznaczyć za pomocą procedury iteracyjnej, która w j-tej iteracji wyznacza w zbiorze A_j, złożonym z elementów, których status jest chwilowo nieokreślony (zwanym zbiorem kandydatów), pewien element maksymalny, usuwa ten element z A_j i dołącza ten element do zbioru uprzednio wyznaczonych elementów maksymalnych N_j. Na początku $N_0 = \emptyset$, $A_0 = Z$. W każdej iteracji $N_j \supset N_{j-1}$, $A_j \subset A_{j-1}$ i w obu przypadkach są to inkluzje właściwe. Z powyższego faktu, a także z faktu, że zbiór Z jest zbiorem skończonym, wynika że algorym kończy pracę po skończonej liczbie iteracji. W powyższym schemacie pojednyncza iteracja odpowiada wyznaczeniu jednego elementu maksymalnego. Element maksymalny wyznaczony w iteracji j-tej ustala status co najmniej jednego elementu A_j. Mianowicie, ustala on swój własny status oraz ewentualnie także status innych elementów ze zbioru kandydatów, ponieważ wszystkie elementy mniejsze niż pewien element maksymalny mogą być usunięte z tego zbioru. Kluczem do efektywności algorytmu jest kolejność wybierania elementów maksymalnych w kolejnych iteracjach zapewniająca możliwie szybką redukcję zbioru kandydatów na wczesnych iteracjach.

Dla przejrzystości w poniższym opisie algorytmu opuścimy indeksy iteracji. Zapis i **mod** k oznacza funkcję, której wartością jest wartość argumentu i **modulo** k.

Algorytm *VM1*

$$A := Z\ ;\ N := \emptyset\ ;\ i := 0\ ;\ STRAT := x\ ;\ x \in \{0,1,\ldots,k\}\ .$$

CYCLE

If $STRAT = 0$ **then** $i := i$ **mod** $k + 1$ **else** $i := STRAT$.
{Wyznaczenie elementu maksymalnego w zbiorze Z}
$m := i$; $t := i$;

SMALL CYCLE

Wyznacz w zbiorze A element, którego m-ta współrzędna osiąga wartość maksymalną y_m^* w zbiorze $A(m) = \{ y \in A \mid y_l = y_l^*, \ldots, \quad l = s \bmod k + 1, \quad s = i - 1, i, \ldots, t - 2 \}$. Jeżeli istnieje tylko jeden taki element, przerwij działanie pętli. Jeżeli takich elementów jest więcej niż jeden, to $m := (m + 1) \bmod k$; $t := t + 1$; oraz **repeat SMALL CYCLE until** $m = i$.

END SMALL CYCLE

Wyznaczony został element maksymalny zbioru Z. Oznaczmy ten element przez $\hat{y}$.
{Aktualizacja zbiorów N i A}
$N := N \cup \{ \hat{y} \}$;
$A := A \setminus (D \cup \{ \hat{y} \})$;
gdzie D jest zbiorem wszystkich elementów A mniejszych niż element maksymalny $\hat{y}$.

END CYCLE

repeat CYCLE.
{Test stopu}
repeat CYCLE until $A = \emptyset$.

END Algorytm *VM1*

W każdej iteracji, to jest za każdym wykonaniem pętli **CYCLE**, algorytm wyznacza jeden element maksymalny poszukując ele-

mentu o maksymalnej wartości i-tej współrzędnej, $i \in \{1,...,k\}$. Jeżeli $STRAT = 0$, to wartość i jest zmieniana cyklicznie w trakcie obliczeń poprzez obliczanie wartości i modulo k i zwiększanie rezultatu o 1 poczynając od 1, tj. $1,...,k,1,...$, tak jak to zostało zaproponowane w pracy [7]. Jeżeli $STRAT \neq 0$, to $i := STRAT$; wartość i jest wtedy stała we wszystkich iteracjach. W tym przypadku algorytm redukuje się do algorytmu zaproponowanego w pracy [8].

Ze schematu algorytmu w oczywisty sposób wynika, że złożoność algorytmu wynosi $O(|Z||N|)$.

Aby poradzić sobie z problemem powtarzania się maksymalnej wartości tej samej współrzędnej dla więcej niż jednego elementu, algorytm korzysta z pętli **SMALL CYCLE**. Jeżeli więcej niż jeden element zbioru Z przyjmuje dla h kolejnych współrzędnych wartość maksymalną, to dla wszystkich tych elementów poszukuje się wartości maksymalnej dla następnej ((**modulo** k) + 1) współrzędnej. Po najwyżej k iteracjach pętli **SMALL CYCLE** wyznaczany jest pewien element maksymalny.

Dla zilustrowania obliczeń w pętli **SMALL CYCLE** załóżmy, że $k = 5$, należy wykonać tę pętlę pięciokrotnie oraz, że na początku tej pętli mamy następujące wartości $m = t = i = 3$. Wówczas w czasie obliczeń w pętli wartości zmiennych m, t, s, oraz l przyjmują następujące wartości:

$$\begin{array}{lll}
m = 3, & t = 3, & s \in \varnothing, \quad l \in \varnothing; \\
m = 4, & t = 4, & s = 2, \quad l = 3; \\
m = 5, & t = 5, & s = 2,3, \quad l = 3,4; \\
m = 1, & t = 6, & s = 2,3,4, \quad l = 3,4,5; \\
m = 2, & t = 7, & s = 2,3,4,5, \quad l = 3,4,5,1.
\end{array}$$

Pozostałe elementy algorytmu są oczywiste i nie wymagają objaśnienia.

Rezultaty numerycznych testów przeprowadzonych z powyższym algorytmem zostały przedstawione w pracach [6] i [7].

4.4. Algorytm równoległy

Algorytm równoległy, który obecnie przedstawimy jest w zasadzie przedstawionym powyżej algorytmem sekwencyjnym, w którym pętla **CYCLE** została zdekomponowana pomiędzy k wzajemnie komunikujące się procesy.

Algorytm *VMa*

$A := Z$; $N := \varnothing$;

CYCLE

{Równoległe wyznaczanie elementu maksymalnego w zbiorze Z }
Zainicjalizuj k procesy.

PARALLEL

W procesie i-tym ($i = 1,\ldots, k$) $STRAT := i$. Każdy proces wyznacza w swojej kopii zbioru kandydatów A element maksymalny wykorzystując algorytm VM1 oraz właściwą dla danego procesu wartość $STRAT$. Niech $\hat{y}^i$ oznacza element maksymalny wyznaczony przez i-ty proces.

{Aktualizacja zbiorów N i A }
Z procesu i-tego prześlij do pozostałych $k-1$ procesów element $\hat{y}^i$ oraz przyjmij elementy $\hat{y}^j$, $j = 1,\ldots,k$, $j \neq i$. $N := N \cup \{ \hat{y}^j \}$; $A := A \backslash \{ D \cup (j \bigcup_j \{ \hat{y}^j \}))$; $j = 1,\ldots,k$, gdzie D jest zbiorem wszystkich elementów mniejszych niż którykolwiek element $\hat{y}^j$, $j = 1,\ldots,k$.

END PARALLEL

repeat CYCLE .
{Test stopu}
repeat CYCLE until $A = \varnothing$.

END Algorytm *VMa*

Zwróćmy uwagę, że każdy proces posiada własną kopię zbiorów A i N. Zwróćmy także uwagę na fakt, że jedna iteracja algorytmu *VMa* jest równoważna wykonaniu k iteracji algorytmu *VM*1 dla $STRAT = 0$.

Przewaga algorytmu równoległego nad algorytmem sekwencyjnym wydaje się oczywista. W każdej iteracji wyznaczanych jest k nowych elementów maksymalnych i są one następnie używane do redukcji zbioru kandydatów. Można zatem spodziewać się, że (teoretycznie) algorytm równoległy przyśpieszy obliczenia k razy oraz oczekiwać w praktyce przyśpieszenia bliskiego tej wartości. Dokładniej, ograniczenie na przyśpieszenie wynosi $|N| / \lceil |N| / k \rceil$, gdzie k jest liczbą procesów, natomiast $\lceil x \rceil$ oznacza najmniejszą liczbę całkowitą większą lub równą x.

Przyśpieszenie (teoretyczne) może jednak nie być osiągane, jeżeli w kolejnych iteracjach liczba *różnych* elementów maksymalnych będzie mniejsza niż k. Efekt ten jest całkowicie zależny od danych i w przypadkach krańcowych może on być znaczny. Zwróćmy także uwagę na fakt, że równoległe poszukiwanie elementów maksymalnych jest jedynym fragmentem algorytmu VMa odpowiedzialnym za przewagę algorytmu równoległego nad algorytmem sekwencyjnym. Po wyznaczeniu w danej iteracji elementów maksymalnych i rozesłaniu ich do wszystkich procesów, każdy proces we własnym zakresie wykonuje uaktualnienie zbiorów A oraz N. Powstaje zatem pytanie, czy zysk wynikający z równoległego wyznaczenia k elementów maksymalnych (co powoduje szybszą redukcję zbioru kandydatów) będzie w stanie zrekompensować czas komunikacji pomiędzy procesorami. Co więcej, wszystkie procesy podlegają zsynchronizowaniu w fazie wymiany informacji: zarówno nadawca jak i odbiorca muszą być gotowi do nadania i przyjęcia informacji. Ponieważ procesy są do pewnego stopnia sterowane poprzez dane konkretnego zadania (w niektórych procesach więcej niż jeden

element może osiągać dla przeszukiwanej współrzędnej wartość maksymalną i z tego powodu procesy te wykonywać będą więcej obliczeń niż inne procesy), mogą one być gotowe do rozpoczęcia wymiany informacji w różnym chwilach czasu. Zatem w każdej iteracji proces "najwolniejszy" określa chwilę rozpoczęcia wymiany komunikacji. Odpowiedź na powyżej sformułowane pytanie można otrzymać jedynie za pomocą eksperymentów numerycznych.

4.5. Eksperymenty numeryczne

Wykorzystywana w eksperymentach sieć transputerów została opisana w Rozdziale 2. Składa sie ona z dwóch transputerów: T4 (korzeń sieci) i T8 (transputer wolny).

Algorytmy VM1 i VMa zostały zaimplementowane w języku Parallel C. "Prostokątne" zadania testowe (P) wygenerowane zostały poprzez równomierne losowanie liczb całkowitych z przedziału [1,9999]. W ten sam sposób wygenerowano zadania "kuliste" (K) , z tym, że w tym przypadku każdy element x zbioru Z spełnia dodatkowy warunek $\sum_{i=1}^{k} (x_i)^2 \le (9999)^2$. Wymiar zadań (to jest wartość k) wynosi 2 . Każda grupa zadań testowych zawiera 6 zadań o różnej liczbie elementów.

Dla umożliwienia poprawnego porównywania pomiędzy algorytmami VM1 i VMa, w algorytmie VM1 parametr *STRAT* musi być równy 0. Rezultaty eksperymentów zostały przedstawione w tabelach 4.1-4.7. Wszystkie czasy, z wyjątkiem tabeli 4.5, podane są w sekundach.

Tabela 4.1. - Algorytm VM1: Zadania P.

Zadanie	$\|Z\|$	$\|N\|$	t_{sz}
1	500	7	0,111
2	1000	10	0,283
3	2000	6	0,330
4	5000	6	0,957
5	10000	5	1,738
6	20000	7	4,247

Tabela 4.2. - Algorytm VM1: Zadania K.

Zadanie	$\|Z\|$	$\|N\|$	t_{sz}
7	500	24	0,351
8	1000	26	0,785
9	2000	34	2,011
10	5000	39	5,896
11	10000	40	13,676
12	20000	40	28,776

Tabela 4.3. - Algorytm VMa: Zadania P.

Zadanie	$\|Z\|$	$\|N\|$	t_{dt}	t_r	S
1	500	7	0,027	0,085	1,31
2	1000	10	0,054	0,201	1,41
3	2000	6	0,108	0,253	1,30
4	5000	6	0,267	0,774	1,24
5	10000	5	0,536	1,392	1,25
6	20000	7	1,083	3,063	1,34

Tabela 4.4. - Algorytm VMa: Zadania K.

Zadanie	$\|Z\|$	$\|N\|$	t_{dt}	t_r	S
7	500	24	0,027	0,252	1,39
8	1000	26	0,054	0,549	1,43
9	2000	34	0,108	1,405	1,43
10	5000	39	0,267	4,331	1,36
11	10000	40	0,536	9,496	1,44
12	20000	40	1,083	19,432	1,48

Tabela 4.1 i tabela 4.2 zawierają rezultaty eksperymentu dla algorytmu *VM1*. Symbol t_{sz} oznacza czas obliczeń. Obliczenia prowadzone były na transputerze T4.

Tabela 4.3 i tabela 4.4 zawierają rezultaty eksperymentu dla algorytmu *VMa*. W eksperymentach z tym algorytmem oba procesy przypisane zostały do osobnych transputerów. Każdy przebieg programu rozpoczyna się wygenerowaniem w korzeniu sieci zadania o odpowiednim wymiarze, a następnie przesłaniu tego zadania do transputera wolnego. W obu tabelach t_{dt} oznacza czas przesyłania danych pomiędzy transputerami, t_r oznacza czas obliczeń (t_{dt} nie jest wliczony do t_r), oraz s oznacza przyśpieszenie algorytmu *VMa* względem *VM1* obliczone według wzoru:

$$s = \frac{\text{czas obliczeń dla } VM1}{\text{czas obliczeń dla } VMa} = \frac{t_{sz}}{t_r}.$$

Na podstawie uzyskanych rezultatów widzimy, że odpowiedź na postawione w paragrafie 4 pytanie jest pozytywna: dla wszystkich zadań przyśpieszenie wynosi więcej niż jeden. Jednak z faktu, że nie przekracza ono nigdy wartości 1.5, możemy wnioskować, że synchronizowanie procesów w fazie komunikacji spowalnia w sposób istotny obliczenia. Aby móc głębiej przyjrzeć się temu zjawisku,

został wykonany prosty test. Dla arbitralnie wybranego zadania 7 zmierzony został czas wykonania pojedynczych iteracji. Uzyskano następujące rezultaty (tabela 4.5 : czasy w 10^{-6} sec).

Tabela 4.5.

Iter.	Czas	Iter.	Czas	Iter.	Czas
1	30912 *	2	18752	3	26816 *
5	20922 *	6	15616	7	17728
9	14976 *	10	11392	11	12922
13	11456	14	14272 *	15	10880 *
17	9856 *	18	8960	19	9152 *
21	9024 *	22	8320	23	8000 *

Suma czasów wykonania wszystkich iteracji wynosi 339572. W przypadku, gdy to samo zadanie będzie rozwiązywane za pomocą algorytmu *VMa* oraz dwóch procesów, to w pierwszej iteracji tego algorytmu pierwszy proces będzie musiał czekać na komunikację z drugim procesem 30912 - 18752 = 12160 mikrosekund. Istotne różnice w czasach obliczeń dla poszczególnych procesów możemy również zaobserwować w pozostałych iteracjach. Wykorzystując powyżej przedstawiony test możemy określić teoretyczną granicę na przyśpieszenie dla wybranego przez nas zadania w sposób następujący. Podzielimy wszystkie iteracje na pary: (1-2), (3-4), etc, i dla każdej pary wybierzemy większy z czasów obliczeń. Czasy te zostały oznaczone w tablicy gwiazdką. Suma wszystkich oznaczonych czasów wynosi 191354 mikrosekund. Czas ten jest dolnym ograniczeniem na czas potrzebny algorytmowi *VMa* na rozwiązanie zadania 7. Wynika z tego ograniczenie na przyśpieszenie, które w rozpatrywanym przypadku nie może być większe niż 339572 : 91354 = 1,77 . W praktyce algorytm *VMa* jest spowalniany poprzez czas komunikacji pomiędzy procesorami oraz dodatkowymi czasami związanymi z organizacją obliczeń. Różnica

pomiędzy 1,77 a 1,39 (rzeczywiste przyśpieszenie uzyskane dla tego zadania) jest miarą tego spowolnienia. Poprawę tego stanu rzeczy można uzyskać poprzez osłabienie wymagania synchronizacji wymiany informacji. W języku ParallelC istnieje możliwość kreowania równoległych *podprocesów* (zwanych *wątkami*) które mogą być wykorzystane do zorganizowania asynchronicznej wymiany informacji pomiędzy procesami. Można to osiągnąć tworząc pewien podproces, którego zadaniem będzie stała kontrola, czy oczekiwane informacje są już dostępne, oraz sygnalizowanie takiego zdarzenia.

4.6. Równoległy algorytm dekompozycyjny

Zasadniczo różny od algorytmu *VMa* algorytm równoległy otrzymamy przez podzielenie zbioru elementów na k podzbiorów oraz

- w fazie pierwszej:
  wyznaczenie w każdym podzbiorze (równolegle) elementów maksymalnych, a następnie utworzenie sumy teoriomnogościowej tych podzbiorów,
- w fazie drugiej:
  wyeliminowanie z tak utworzonego zbioru tych elementów, które nie są maksymalne.

Przedstawiony powyżej algorytm dekompozycyjny będziemy oznaczać *VMb*. W algorytmie tym komunikacja pomiędzy procesami organiczona jest do minimum. Ze względu na swoją prostotę algorytm ten nie wymaga formalnego opisu. Również i ten algorytm został zrealizowany w języku Parallel C, a następnie przetestowany na tej samej sieci transputerów co algorytm *VMa*. Sposób dekompozycji zbioru elementów jest tutaj rzeczą bardzo istotną. Dla $k = 2$ przyjęta przez nas metoda polega na wykorzystaniu linii separującej $x_1 = x_2$; do pierwszego podzbioru należą elementy takie, że $x_1 \geq x_2$, do drugiego elementy takie, że $x_1 < x_2$. Metoda ta

wydaje się najlepsza dla podziału zbioru elementów maksymalnych na dwa podzbiory o podobnej mocy.

4.7. Eksperymenty numeryczne z równoległym algorytmem dekompozycyjnym

W przeprowadzonych eksperymentach obliczenia w każdym podzbiorze były wykonywane za pomocą algorytmu *VM1* i stanowiły osobny proces. Podobnie jak w przypadku algorytmu *VMa*, każdy proces przypisany został do osobnego transputera. Każdy przebieg algorytmu rozpoczyna się wygenerowaniem w korzeniu sieci zadania o odpowiednim rozmiarze, a następnie przesłaniu jednego z podzbiorów do transputera wolnego.

Tabela 4.6 i tabela 4.7 zawierają rezultaty testowania algorytmu *VMb*. Liczby w nawiasach oznaczają: w kolumnie $|Z|$ - liczbę elementów, które pozostały w korzeniu sieci oraz liczbę elementów przesłanych do transputera wolnego, w kolumnie $|N|$ - liczbę elementów maksymalnych wyznaczonych w fazie pierwszej w korzeniu sieci i na transputerze wolnym. Symbol t_{dt1} oznacza czas transmisji danych z korzenia do transputera wolnego, t_{dt2} czas transmisji danych z transputera wolnego do korzenia, t_1 czas obliczeń w fazie pierwszej, t_2 czas obliczeń w fazie drugiej algorytmu, natomiat s oznacza przyśpieszenie obliczone w ten sam sposób jak dla algorytmu *VM1*. Za łączny czas obliczeń przyjmujemy: $t_r = t_1 + t_2 + t_{dt2}$ pomijając, podobnie jak w przypadku algorytmu *VMa*, czas t_{dt1} .

Tabela 4.6. - Algorytm *VMb* : Zadania P.

zad.	$\|Z\|$	$\|N\|$	t_{dt1}	t_{dt2}	t_1	t_2	s
1	500 (249,251)	7 (7,5)	0,019	0,000	0,068	0,003	1,56
2	1000 (499,501)	10 (10,7)	0,041	0,000	0,185	0,005	1,49
3	2000 (1037,963)	6 (6,10)	0,079	0,000	0,182	0,002	1,79
4	5000 (2630,2370)	6 (6,7)	0,195	0,000	0,456	0,003	2,08
5	10000 (5239,4761)	5 (5,6)	0,389	0,000	0,940	0,002	1,84
6	20000 (10377,9623)	7 (7,4)	0,784	0,000	2,067	0,003	2,05

Tabela 4.7. - Algorytm *VMb* : Zadania K.

zad.	$\|Z\|$	$\|N\|$	t_{dt1}	t_{dt2}	t_1	t_2	s
7	500 (204,296)	24 (17,7)	0,020	0,001	0,140	0,026	2,10
8	1000 (423,577)	26 (19,8)	0,038	0,001	0,342	0,032	2,09
9	2000 (813,1187)	34 (25,9)	0,078	0,001	0,891	0,052	2,13
10	5000 (2016,2984)	39 (26,13)	0,195	0,001	2,266	0,066	2,53
11	10000 (4041,5959)	40 (27,13)	0,391	0,001	5,254	0,070	2,57
12	20000 (8045,11955)	40 (27,13)	0,789	0,001	11,307	0,071	2,44

Zaskakujące jest to, że dla większości zadań otrzymaliśmy przyśpieszenie większe niż 2 . Wynika z tego w sposób natychmiastowy, że algorytm *VM*1 nie jest algorytmem najlepszym. W istocie, wykonując sekwencyjnie algorytm *VMb* otrzymalibyśmy krótsze czasy wykonania niż dla algorytmu *VM*1. Przy okazji testowania algorytmu równoległego *VMa* uzyskaliśmy zatem wskazówki co do konstrukcji algorytmu o efektywności większej niż efektywność algorytmu *VM*1.

4.8. Złożoność algorytmu dekompozycyjnego

Załóżmy, że dokonując podziału zbioru Z na dwie równe części dzielimy również na dwie równe części zbiór N . Stosując dla każdego podzbioru algorytm *VM*1 (o złożoności $O(|Z||N|)$) otrzymujemy sekwencyjny algorytm dekompozycyjny o następującej złożoności:

$$2 \cdot O((|Z|/2) \cdot (|N|/2)) = O(|Z||N|/2).$$

Załóżmy, że potrafimy dokonać podziału zbioru Z na $|N|$ podzbiorów takich, że każdy podzbiór zawiera $|Z|/|N|$ elementów spośród których dokładnie jeden jest maksymalny w Z. Stosując dla każdego podzbioru algorytm *VM*1 otrzymujemy sekwencyjny algorytm dekompozycyjny o złożoności:

$$|N| \cdot O((|Z|/|N|) \cdot (|N|/|N|)) = O(|Z|).$$

Oczywiście, w takim przypadku złożoność równoległego algorytmu dekompozycyjnego wynosi $O(|Z|/|N|)$, a teoretyczne przyśpieszenie w stosunku do sekwencyjnego algorytmu dekompozycyjnego $O(|N|)$.

4.9. Uwagi końcowe

Przedstawione w niniejszym rozdziale implementacje algorytmów stanowią dobrą ilustrację technicznych problemów na jakie natrafiamy w obliczeniach równoległych. Z jednej strony możliwość uzyskania teoretycznego przyśpieszenia obliczeń proporcjonalnego do liczby użytych procesorów jest w wielu zastosowaniach rzeczą bardzo atrakcyjną. Z drugiej jednak strony, jak to mieliśmy okazję zaobserwować, zbliżenie się w praktyce do tej granicy wymaga bardzo szczegółowej analizy algorytmu w odniesieniu do rozwiązywanego problemu, modelu obliczeń i związanej z modelem dysponowanej (lub postulowanej) architektury maszyny równoległej. Zaprezentowane wyniki eksperymentów numerycznych unaoczniają, między innymi, jak łatwo nieodpowiedni dobór tych elementów redukuje rzeczywiste przyśpieszenia do poziomu stawiającego pod znakiem zapytania sensowność stosowania obliczeń równoległych.

Literatura

1. Bitran G.R. (1979): "Theory and Algorithms for Linear Multiple Objective Programs with Zero-one Variables". *Math. Programming*, 17, 362-390.
2. Hoare C.A.R. (1985): Communicating Sequential Processes. Prentice-Hall International, Englewood Cliffs.
3. Evtushenko Yu.G, Potapov M.A. (1986): "Methods of Numerical Solution of Multicriterion Problems". *Soviet Math. Doklady*, 420-423.
4. Kaliszewski I. (1989): "VM - a Parallel Algorithm to Determine Minimal Elements of Discrete Sets". *Prace IBS PAN*, ZPM-16, Warszawa.
5. Kaliszewski I. (1990): "Determination of Maximal Elements of Finite Sets on a Network of Transputers". *Prace IBS PAN*, ZPM-2, Warszawa.
6. Kung H.T., Luccio F., Preparata F.P. (1975): "On Finding the Maxima of a Set of Vectors". *Journal of the ACM*, 22, 469-476.
7. Majchrzak J (1989): "A Methodological Guide to the Decision Support System DISCRET for Discrete Alternatives Problems". W: Aspiration Based Decision Support Systems - Theory, Software and Applications, red. A.Lewandowski, A.P.Wierzbicki, Lecture Notes in Economics and Mathematical Systems, 331, Springer Verlag, Berlin, 193-212.
8. Polak E., Payne A.N. (1976): "On Multicriteria Optimization". w: Directions in Large-Scale Systems, red, Ho and Mitter, Plenum Press, New York and London.
9. Villareal B., Karwan M.H. (1982): "Multiple Dynamic Programming with Applications to the Integer Case". *Journal of Optimization Theory and Applications*, 38, 43-69.

Dodatek 2

RAPORTY I PUBLIKACJE ZAKŁADU PROGRAMOWANIA MATEMATYCZNEGO IBS PAN DOTYCZĄCE RÓWNOLEGŁEJ OPTYMALIZACJI DYSKRETNEJ (1986-92)

I. Publikacje

1. Bertocchi M., Brandolini L., Słomiński L., Sobczyńska J. (1992): "A Monte-Carlo Approach for 0-1 Programming Problems". *Computing* 48, 259-274.
2. Dudziński K. (1986): "Wybrane równoległe algorytmy kombinatoryczne". *Roczniki PTM Seria III, Matematyka Stosowana XXVIII*, 91-123.
3. Kaliszewski I., Wojtowicz M. (1991) "Modest. A language for parallel programming". *Prace IPI PAN*, 691, Warszawa.
4. Słomiński L., Kaliszewski I. (1988): "Problemy obliczeń równoległych". *Prace IBS PAN*, 168, Warszawa.
5. Słomiński L. (1988): "Algorytmy równoległe znajdowania minimaksowych rozgałęzień". *Zeszyty Naukowe Politechniki Śląskiej, ser. Automatyka* 94, 287-301.
6. Słomiński L. (1988): "Obliczenia równoległe i optymalizacja". W: Materiały Konferencji Naukowej: Problemy współczesnej radiolokacji, WAT, Warszawa, 124-130.
7. Słomiński L. (1989): "Parallel Algorithms for Optimization". W: Proceedings of the 3rd Polish-Finish Symp. on Methodology and Applications of Decision Support Systems, red. R. Kulikowski, IBS PAN, Warszawa, 219-233.
8. Słomiński L. (1990): "Implementacje algorytmów kombinatorycznych na symulowanych sieciach wielomikroprocesorowych".

Zeszyty Naukowe Politechniki Śląskiej, ser. Automatyka, 100, 287-300.

9. Słomiński L. (1990): "Parallel Computing for Flexible Manufactoring Systems". W: Decision Making Models for Management and Manufacturing, red. R. Kulikowski, Omnitech Press, Warszawa, 215-229.
10. Słomiński L. (1990): "Komputery równoległe dla rozwiązywania zadań optymalizacji dyskretnej i sztucznej inteligencji". Materiały konferencji "*Sztuczna inteligencja w zarządzaniu*", Warszawa, listopad 1989, PTBiOS i IBS PAN, red. A.Straszak, Z. Nahorski, C. Iwański, Warszawa, 201-210.

II. Raporty (latami, wg alfabetu)

1986

1. Dudziński K.: "Obliczenia równoległe: Wybrane algorytmy kombinatoryczne". ZPM 20/86, IBS PAN, Warszawa.
2. Grygiel G.: "Obliczenia równoległe: Algorytm równoległy dla zadań przydziału". ZPM 24/86, IBS PAN, Warszawa.
3. Kaliszewski I.: "Obliczenia równoległe: Modele obliczeń w języku Ameba-Pascal". ZPM 23/86, IBS PAN, Warszawa.
4. Kaliszewski I., Słomiński L.: "Obliczenia równoległe: przegląd zagadnień". ZPM 17/86, IBS PAN, Warszawa.
5. Majchrzak J.: "Obliczenia równoległe: uwarunkowania i prognozy". ZPM 21/86, IBS PAN, Warszawa.
6. Słomiński L.: "Obliczenia równoległe: algorytm równoległy wyznaczania dendrytu minimaksowego w grafie skierowanym". ZPM 19/86, IBS PAN, Warszawa.

7. Waluk B.: "Obliczenia równoległe: Zastosowanie niektórych metod dekompozycji macierzy współczynników do rozwiązywania układów równań". ZPM 22/86, IBS PAN, Warszawa.

1987

1. Grygiel G.: "Zadanie przydziału: algorytm równoległy i eksperymentalna ocena liczby iteracji". ZPM 34/87, IBS PAN, Warszawa.
2. Kaliszewski I.: "Parallel counterparts of algorithms for determining minimal elements of discrete sets". ZPM 30/87, IBS PAN, Warszawa.
3. Kaliszewski I., Wojtowicz M.: "Język Modest". ZPM 31/87, IBS PAN, Warszawa.
4. Kulczycka D.: "Równoległe algorytmy b-skojarzeń w drzewach". ZPM 23/87, IBS PAN, Warszawa.
5. Majchrzak J., Skawiński A.: "Sieci komputerowe - przegląd literatury i ocena możliwości przetwarzania równoległego". ZPM 17/87, IBS PAN, Warszawa.
6. Majchrzak J., Skawiński A.: "Pakiet PPLANEUP do organizacji przetwarzania równoległego w sieci D-Link komputerów IBM PC/XT/AT". ZPM 18/87, IBS PAN, Warszawa.
7. Majchrzak J., Skawiński A.: "Propozycja bezgradientowej metody równoległej optymalizacji nieliniowej bez ograniczeń". ZPM 19/87, IBS PAN, Warszawa.
8. Słomiński L.: "Narzędzia optymalizacji równoległej na maszynach sekwencyjnych". ZPM 13/87, IBS PAN, Warszawa.
9. Słomiński L.: "Problemy równoległych algorytmów dla przepływu maksymalnego w sieci". ZPM 14/87, IBS PAN, Warszawa.

1988

1. Gondek H., Słomiński L.: "Implementacja na symulowanej maszynie równoległej typu dwudrzewo algorytmu Dijkstry wyznaczania drzewa dróg najkrótszych w digrafie". ZPM 37/88, IBS PAN, Warszawa.
2. Grygiel G., Kulczycka D.: "Ocena systemu symulacyjnego JUPITER-86". ZPM 24/88, IBS PAN, Warszawa.
3. Grygiel G.: "Symulacja algorytmu równoległego dla zadania przydziału przy pomocy pakietu JUPITER-86". ZPM 44/88, IBS PAN, Warszawa.
4. Kaliszewski I., Kulczycka D., Słomiński L.: "Równoległy algorytm Prima-Dijkstry wyznaczania dendrytu minimalnego: implementacja na symulowanej strukturze typu dwudrzewo". ZPM 5/88, IBS PAN, Warszawa.
5. Kaliszewski I., Słomiński L.: "Problemy obliczeń równoległych". ZPM 38/88, IBS PAN, Warszawa.
6. Kulczycka D.: "Algorytmy równoległe b-skojarzeń w drzewach: implementacja w systemie symulacyjnym Jupiter". ZPM 6/88, IBS PAN, Warszawa.
7. Majchrzak J.: "Pakiet PPLANEUP wersja 2 dla wspomagania przetwarzania równoległego w sieciach D-Link komputerów IBM PC/XT/AT". ZPM 41/88, IBS PAN, Warszawa.
8. Słomiński L.: "Algorytmy równoległe znajdowania minimaksowych rozgałęzień". ZPM 1/88, IBS PAN, Warszawa.
9. Słomiński L.: "Obliczenia równoległe i optymalizacja". ZPM 4/88, IBS PAN, Warszawa.

1989

1. Gondek H., Słomiński L.: "Symulowane odmiany struktury dwudrzewa dla algorytmów najkrótszych dróg w digrafie". ZPM 10/89, IBS PAN, Warszawa.

2. Gondek H.: "Równoległy algorytm mnożenia macierzy. Implementacja i eksperyment obliczeniowy na symulowanej strukturze typu mesh i hypercube o n^2 procesorach". ZPM 11/89, IBS PAN, Warszawa.
3. Grygiel G.: "Równoległość na poziomie podprocedur na przykładzie zadania przydziału". ZPM 13/89, IBS PAN, Warszawa.
4. Grygiel G.: "Program AZP rozwiązujący algebraiczne zagadnienie przydziału - opis funkcjonalny". ZPM 18/89, IBS PAN, Warszawa.
5. Kaliszewski I.: "VM1 - A parallel algorithms to determine minimal elements of discrete sets". ZPM 16/89, IBS PAN, Warszawa.
6. Kulczycka D.: "Algorytmy równoległe w sieci drzew ortogonalnych - realizacja w systemie symulacyjnym JUPITER-86". ZPM 12/89, IBS PAN, Warszawa.
7. Majchrzak J.: "O programowaniu dla transputerów INMOS T800". ZPM 14/89, IBS PAN, Warszawa.
8. Słomiński L.: "Parallel Algorithms for Optimization". ZPM 1/89, IBS PAN, Warszawa.
9. Słomiński L.: "Komputery równoległe dla rozwiązywania zadań optymalizacji i sztucznej inteligencji". ZPM 3/89, IBS PAN, Warszawa.
10. Słomiński L., Sobczyńska J.: "Programy i dane w języku Fortran 77 dla wybranych zadań dyskretnych, z uwzględnieniem potrzeb wektoryzacji". ZPM 4/89, IBS PAN, Warszawa.
11. Sobczyńska J.: "Charakteryzacja wieloprocesorowej struktury Perfect Shuffle. Równoległy algorytm mnożenia macierzy i jego implementacja w tej strukturze". ZPM 9/89, IBS PAN, Warszawa.

1990

1. Grygiel G.: "Implementacja asynchronicznego algorytmu dla zadania przydziału". ZPM 26/90, IBS PAN, Warszawa.

2. Kaliszewski I.: "On determining minimal elements of discrete sets on a network of transputers". ZPM 4/90, IBS PAN, Warszawa.
3. Kulczycka D.: "Prosty algorytm równoległy znajdowania skojarzenia maksymalnego". ZPM 18/90, IBS PAN, Warszawa.
4. Majchrzak J.: "Transputer i sieci wielotransputerowe". ZPM 30/90, IBS PAN, Warszawa.
5. Słomiński L.: "Komunikacja w systemach wieloprocesorowych i jej wpływ na efektywność obliczeń". ZPM 22/90, IBS PAN, Warszawa.

1991

1. Bertocchi M., Butti A., Sergi P., Słomiński L., Sobczyńska J.: "Stochastic and Deterministic Optimization on 0-1 Multiknapsack Problem". *Quaderni del Dipartimento di Mathematica, Statistica, Informatica e Applicazioni*, No 14/91, Bergamo.
2. Kulczycka D.: " Algorytm O(|E|) skojarzenia w grafie dwudzielnym". ZPM 13/91, IBS PAN, Warszawa.

1992

1. Bertocchi M., Butti A., Słomiński L.: "Fixed Precision Random Search Procedure for the Binary Multiknapsack Problem". (Revised version). *Quaderni del Dipartimento di Mathematica, Statistica, Informatica e Applicazioni*, No 18/92, Bergamo.
2. Grygiel G., Kabanow P., Słomiński L., Sobczyńska J.: "Wykorzystanie sieci transputerowych do rozwiązania wielowymiarowego zadania załadunku metodą Monte-Carlo". /w języku rosyjskim/ ZPM 12/92, IBS PAN, Warszawa.