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.

6. Modest: język programowania równoległego

6.1. Wstęp

Algorytmy równoległe mają ścisły związek z pojęciem współbieżności procesów obliczeniowych. Proces obliczeniowy jest definiowany jako sekwencja operacji arytmetycznych. W maszynie sekwencyjnej pracującej w systemie wielozadaniowym procesy (zadania) są obsługiwane przez jednostkę centralną cyklicznie w określonych przedziałach czasu (time-sharing) tak, że użytkownicy mają wrażenie jednoczesnej obsługi zadań przez system. W takim przypadku mówimy, że procesy przetwarzane są współbieżnie.

Współbieżność jest pojęciem kluczowym dla efektywnej organizacji obliczeń. Naturalnym sposobem zwiększenia efektywności obliczeń jest zwiększenie liczby procesorów i rozdzielenie procesów współbieżnych na procesory. Mówimy wtedy, że procesy współbieżne przetwarzane są równolegle. Zatem procesy równoległe są współbieżne ale nie odwrotnie.

Algorytmy numeryczne różnią się od innych zadań przetwarzanych w systemie operacyjnym tym, że głównym, a często i jedynym kryterium ich jakości jest czas wykonania. Zasada współbieżności w zastosowaniu do algorytmów numerycznych rozumiana jako możliwość rozbicia algorytmu na fragmenty, które mogą być przetwarzane współbieżnie, jest zatem interesująca o tyle, o ile fragmenty te mogą być przetwarzane równolegle; tylko wtedy możemy się spodziewać przyśpieszenia czasu wykonywania algorytmu.

Jeżeli mamy do dyspozycji tyle procesorów ile fragmentów algorytmu, to mówimy o nieograniczonej równoległości. Założenie o nieograniczonej równoległości występuje nagminnie w najczęściej stosowanych modelach obliczeń równoległych. Jeżeli jednak założenie to nie jest spełnione, co oznacza, że jesteśmy zmuszeni

pracować w warunkach ograniczonej równoległości, możemy zainicjalizować tyle fragmentów ile mamy dostępnych procesorów. Może się jednak zdarzyć, że korzystnne jest wykonywanie wszystkich fragmentów stopniowo (na przykład, poszczególne fragmenty mogą być źródłem informacji, które mogą przyśpieszyć obliczenia) i wtedy obliczenia należy prowadzić w trybie równoległo-współbieżnym. Modest, język który tutaj przedstawiamy, pozwala radzić sobie z ograniczoną równoległością (która jest naturalną cechą rzeczywistych maszyn równoległych) poprzez możliwość wyboru pomiędzy równoległym i równoległo-współbieżnym trybem obliczeń. Wymaga to jednak aby przy definiowaniu języka posługiwać się, oprócz pojęcia *proces*, także pojęciem *procesor* (*liczba procesorów*), które pozwala przenieść na grunt języka kwestię fizycznych zasobów obliczeń równoległych. Inne cechy języka Modest są typowe dla przyjaznego użytkownikowi języka obliczeń równoległych. Mianowicie:

- język pozwala na łatwe modelowanie wzajemnych związków pomiędzy procesami,
- semantyka języka jest prosta i dzięki temu można konstruować programy, które w dużym stopniu odzwierciedlają podstawowe cechy konkretnej maszyny,
- język jest rozszerzeniem języka Pascal, dzięki czemu opanowanie jego jest bardzo łatwe.

Chociaż konstruowanie efektywnych algorytmów dla obliczeń równoległych jest procesem złożonym, wymagającym sprawności w programowaniu oraz głebokiego wniknięcia w rozwiązywany problem i wybrany model obliczeń, Modest może się stać narzędziem programisty w krótkim czasie. Nie wymaga poznania nowego języka od podstaw, a jedynie formalizuje, w odniesieniu do obliczeń równoległych, naturalne intuicje programisty przyzwyczajonego do "sekwencyjnego" języka Pascal.

W odróżnieniu od algorytmów sekwencyjnych, dla których obowiązuje jeden podstawowy model obliczeń, istnieje wiele modeli obliczeń równoległych wynikających z różnorodności architektur maszyn równoległych. Wydaje się to mało prawdopodobne, aby kiedyś w przyszłości powstał jeden ogólny model takich obliczeń. Intencją autorów było stworzenie języka, który mógłby być wykorzystany dla zapisania dowolnego algorytmu w dowolnym modelu obliczeń równoległych, w którym obliczenia sterowane są strumieniem danych, co ustaliłoby wspólne ramy dla porównań i eksperymentów (przy braku dostępu do maszyn wieloprocesorowych eksperymenty mogłyby być prowadzone drogą symulacji). Tłumaczy to dlaczego Modest nie posiada szczegółowych mechanizmów kontroli dostępu do wspólnych zasobów, które muszą być stosowane w modelach obliczeń równoległych ze wspólną pamięcią, a które są dostępne językach programowania współbieżnego takich jak Concurrent Pascal, Modula, czy Ada. W języku Modest zbudowanie takich mechanizmów pozostaje zadaniem programisty i może to być odbierane przez niektórych użytkowników języka jako jego ułomność. Autorzy jednak wierzą, że inne cechy języka, między innymi jego prostota, uczynią go atrakcyjnym dla szerokiego grona programistów i użytecznym dla rozległej gamy zastosowań.

Główną motywacją stojącą zastworzeniem języka Modest była chęć posiadania użytecznego narzędzia dla opisu algorytmów równoległych. Jak dotąd język ten nie posiada implementacji.

6.2. Opis języka

Wszystkie pojęcia, które nie są tutaj jawnie definiowane, są pojęciami pochodzącymi z języka Pascal (por. [1,4]).

6.2.1. Deklaracje typu

```
typ = typ_zbiorowy | typ_komunikacyjny | inny_typ.
```

```
typ_zbiorowy = set of inny_typ.
typ komunikacyjny = room for inny_typ.
```

W powyższych definicjach każdy typ różny od typu zbiorowego i typu komunikacyjnego (to znaczy typ pascalowy) jest określany jako inny_typ. Powyższe definicje dopuszczają, pomiędzy innymi, następujące typy:

```
I = SET OF INTEGER ;
R = SET OF REAL ;
S = SET OF ARRAY [1..10] OF REAL ;
```

Można zatem korzystać w języku Modest ze zbiorów elementów dowolnego typu pascalowego. Jest to rozszerzenie definicji typu zbiorowego używanego w języku Pascal.

Typ komunikacyjny został wprowadzony dla wyróżnienia zmiennych wykorzystywanych dla organizacji komunikacji pomiędzy współbieżnymi procesami.

6.2.2. Zmienne

Wymaga się aby każda zmienna była zainicjowana przez program. Oznacza to, że moc dowolnego zbioru reprezentowanego przez zmienną typu zbiorowego jest zawsze skończona. Dla zmiennej o typie innym niż prosty wymaga się aby wszystkie elementy zmiennej były zainicjowane.

Zmienne komunikacyjne mają typ komunikacyjny i są używane dla wymiany danych pomiędzy procesami. Zmienne te są parametrami dla procedur dla składowania danych i dostępu do danych. Z natury tych zmiennych wynika, że nie są one lokalne. Dwa (lub więcej) procesy mają dostęp to tej samej zmiennej komunikacyjnej jeżeli zmienne te zostały zadeklarowane w odpowiednich procedurach współbieżnych. W ten sposób można zorganizować dowolny

typ komunikacji pomiędzy procesami - począwszy od modelu ze wspólną pamięcia do dowolnej sieci procesorów.

Przykład 1.

```
...
x: X;
c: room for X;
...
{ * dostęp do danych * }
access(x,c);
{ * składowanie danych * }
pile(x,c);
...
```

(por. podrozdział 2.3).

Stan zmiennej komunikacyjnej charakteryzują flaga dostępu oraz liczniki ilości wywołań instrukcji **access** i **pile**. Z definicji instrukcji **access** (por. 2.3) wynika, że jeżeli zmienna komunikacyjna jest niedostępna, to odpowiadający jej proces jest zawieszany aż do chwili kiedy zmienna stanie się dostępna. W niektórych sytuacjach wygodniej jest, zamiast wywoływać bezpośrednio procedurę **access**, sprawdzić stan zmiennej komunikacyjnej i w przypadku, gdy zmienna jest niedostępna kontynuować obliczenia. Definicja flagi zmiennej komunikacyjnej jest następująca:

```
flaga = ?identyfikator_zmiennej_komunikacyjnej.
```

Flaga przyjmuje dwie wartości **true** i **false** w zależności od dostępności zmiennej komunikacyjnej (**true** - dostępna, **false** - niedostępna). Zmienna może być uczyniona niedostępną przez wywołanie procedury **clear** ze zmienną jako parametrem. Po wywoła-

niu **clear** zmienna jest niedostępna (wartość flagi jest **false**) aż do następnej instrukcji **pile**, która nadaje zmiennej nową wartość.

Liczba skutecznych dostępów do zawartości zmiennej oraz liczba składowań liczona od ostatniego wywołania procedury **clear** (lub od początku obliczeń jeżeli procedura **clear** z daną zmienną jako parametrem nie była uprzednio wywoływana) jest udostępniana poprzez funkcje **access?** i **pile?** których argumentami są zmienne komunikacyjne, jak to ilustruje następujący przykład.

Przykład 2.

```
...
var
x: room for real ;
begin
   write('liczba składowań',pile?(x),'liczba dostę-
      pów',access?(x))
end ;
...
```

6.2.3. Instrukcje

W języku Modest oprócz instrukcji pascalowych określone są:

- instrukcje komunikacji pomiędzy procesami,
- instrukcje inicjalizowania procesów.

```
instrukcja_komunikacji = instrukcja_składowania | in-
   strukcja_dostępu.
instrukcja_składowania = pile(parametry_składowania).
instrukcja_dostępu = access(parametry_dostępu).
parametry_składowania = identyfikator_zmiennej_komunika-
   cyjnej, wyrażenie{,wyrażenie}.
parametry_dostępu = identyfikator_zmiennej_komunikacyj-
   nej, identyfikator_zmiennej{,identyfikator_zmiennej}.
```

Instrukcja **access** wykonywana jest tylko wtedy, gdy zmienna komunikacyjna jest dostępna. Jeżeli jest ona niedostępna, to dany proces jest zawieszany aż do chwili kiedy zmienna stanie się dostępna.

```
instrukcja_inicjalizacji_procesu = instrukcja_inicjali-
   zacji_bezwarunkowej | instrukcja_inicjalizacji_warun-
   kowej.
instrukcja_inicjalizacji_bezwarunkowej = parallel wywo-
   łanie_procedury_współbieżnej
{ , wywołanie_procedury_współbieżnej } |
concurrent wywołanie_procedury_współbieżnej
{ , wywołanie_procedury_współbieżnej }.
instrukcja_inicjalizacji_warunkowej = for
identyfikator_zmiennej in
identyfikator_zmiennej_zbiorowej
{ , for identyfikator_ zmiennej in identyfikator_zmien-
   nej_zbiorowej } create
instrukcja_inicjalizacji_bezwarunkowej.
```

Jeżeli w instrukcji inicjalizacji warunkowej słowo kluczowe **for** występuje jednokrotnie, to wszyskie formalne parametry procedury za wyjątkiem jednego muszą być parametrami aktualnymi. Jeżeli nie spowoduje to przekroczenia liczby procesów które mogą być aktywne w programie, to instrukcja zaincjuje k procesów (k jest licznością elementów znajdującą się aktualnie w zbiorze reprezentowanym przez zmienną zbiorową) i każy proces jest wykonaniem procedury, której parametr formalny został podstawiony przez element zbioru. W następstwie tego zbiór staje się pusty. Jeżeli jednak możliwe jest zainicjowanie jedynie l procesów, gdzie l jest mniejsze niż k , to tylko l procesów zostanie zainicjowane a w zbiorz pozostanie $k - l$ elementów.

Podobnie, jeżeli w instrukcji inicjalizacji warunkowej słowo kluczowe **for** występuje m krotnie, to wszyskie formalne parametry

procedury za wyjątkiem *m* parametrów muszą być parametrami aktualnymi.

Przykład 3.

```
...
co-routine A(i : integer) ;
begin
...
end ;
   var i_set : set of integer ; i : integer ;
...
begin
   i_set := [1,2,3,4];
   for i in i_set create parallel A(i)
end ;
...
```

Jeżeli w chwili wykonywania instrukcji będzie można zaincjalizować cztery procesy, to zbiór *i_set* stanie się zbiorem pustym. Jeżeli będzie można zainicjalizować tylko jeden proces, to zbiór *i_set* zawierać będzie trzy elementy, przy czym nie określa się które to będą elementy.

6.2.4. Deklaracje i wywoływanie procedur współbieżnych

Podstawową konstrukcją jezyka Modest pozwalającą na konstruowanie algorytmów równoległych jest procedura współbieżna. Procedura współbieżna, podobnie jak "zwykłe" procedury, jest treścią pewnego procesu obliczeniowego i odwrotnie, proces obliczenowy zostaje zainicjowany przez wywołanie procedury współbieżnej.

```
deklaracja_procedury_współbieżnej = nagłówek-procedury_
   współbieżnej ; blok_procedury współbieżnej | nagłó-
   wek_procedury_współbieżnej ; dyrektywa | identyfika-
```

```
   cja_procedury_współbieżnej ; blok_procedury współ-
   bieżnej.
nagłówek_procedury-współbieżnej = co_routine identyfika-
   tor_procedury_współbieżnej [lista_parametrów_formal-
   nych].
identyfikacja_procedury_współbieżnej = co-routine iden-
   tyfikator_procedury_współbieżnej.
identyfikator_procedury_współbieżnej = identyfikator.
blok_procedury_współbieżnej =
            część_deklarująca_etykiety
            część_deklarująca_stałe
            część_deklarująca_typy
            część_deklarująca_zmienne
            część_specyfikująca_zmienne_nielokalne
            część_deklarująca_procedury_i_funkcje
            część_operacyjna.
część_deklarująca_zmienne_nielokalne = use lista_iden-
   tyfikatorów_zmiennych.
```

Ponieważ jeden proces może zainicjować kilka procesów oraz ponieważ proces inicjujący może być kontynuowany, parametry procedur współbieżnych mogą być przekazywane jedynie przez wartość lub przez nazwę. Wymaganie wyspecyfikowania zmiennych nielokalnych wynika z dążenia do dyscyplinowania procesu konstruowania programów równoległych.

Przykład 4.

```
co_routine sqrt(x:real) ;
begin
   if x ≥ 0 then k:=k+1 else y:=y+sqrt(x)
end ;
```

W powyższym przykładzie zmienne k, y są nielokalne. Mechanizm dostępu do zmiennych nielokalnych, w przypadku gdy zmienne te występują w różnych procesach, musi być zaprojektowany i zaprogramowany przez programistów.

Przykład 5.

```
co_routine transfer(i : integer) ;
type a : room for integer ;
var j,l : integer; chain : array [1..n] of a ;
procedure xt(var k:integer) ;
begin
   ...
   k :=... ;
   ...
end
begin
   j:=i ;
   while j ≥ 0 begin
      xt(j) ;
      pile(chain[(i+1)mod n],j) ;
      access(chain[(i-1)mod n],j)
      end
   pile(chain[(i+1)mod n],j)
end ;
```

Proces *transfer* (i) , $i = 0, 1, \ldots, n-1$, którego treścią jest procedura współbieżna *transfer* , przesyła cyklicznie (pętla while) wynik działania procedury *xt* do procesu *transfer* (($i+1$) **mod** n) i pobiera od procesu *transfer* (($i-1$) **mod** n) wynik działania procedury *xt* . Zainicjowanie n procesów *transfer* z parametrami $i = 0, 1, \ldots, n-1$ powoduje utworzenie pierścienia procesów, w którym informacja krąży w sposób cykliczny. Zakończenie obliczeń w pierścieniu nastąpi, gdy wszystkie procesy osiągną warunek $j < 0$ (przerwanie komunikacji w pierścieniu nastąpi, gdy co najmniej jeden proces osiągnie warunek $j < 0$).

Zwróćmy uwagę na fakt, że ponieważ instrukcja **access** jest pewną formą instukcji **read**, nie jest wiadome czy informacja do której proces uzyskuje dostęp jest informacją nową czy też informacją udostępnioną podczas poprzedniego wykonania instrukcji **access** (procesy nie są synchronizowane). Można to jednak zawsze

ustalić korzystając z informacji dostarczanej przez instrukcję **access?.**

Przykład 6.

```
for i:=1 to n do parallel sqrt ( a [ i ] ) ;
```

Instrukcja ta inicjuje n procesów sqrt($a[i]$) , $i = 1,..., n$. Każdy proces żąda dla siebie osobnego procesora.

Przykład 7.

```
for i:=1 to n do concurrent sqrt ( a [ i ] ) ;
```

Dzialanie tej instrukcji jest analogiczne do działania instrukcji poprzedniej, z tym że w tym przypadku proces inicjujący i zainicjowane n procesów *sqrt* zostaną wykonane współbieżnie na tym samym procesorze.

W obu powyższych przykładach zmienna sterująca instrukcji **for** służy do wskazania parametru aktualnego procedury w instrukcji bezwarunkowej inicjalizacji procesu.

Przykład 8.

```
for a in A create concurrent sqrt(a,b) ;
```

Rezultatem wykonania tej warunkowej instrukcji inicjalizacji procesu jest wykreowanie ciągu procesów sqrt o liczności równej ilości elementów skonkretyzowanych zmiennej A mającej typ zbiorowy. Zmienna sterująca a musi mieć typ zgodny z typem elementów A . Wszystkie zainicjowane procesy zostaną wykonane współbieżnie na tym samym procesorze.

Przykład 9.

```
for a in A create parallel sqrt(a,b) ;
```

Instrukcja ta ma działanie analogiczne do instrukcji z poprzedniego przykładu, z tym, że w tym przypadku każdy proces żąda dla siebie oddzielnego procesora.

6.2.5. Funkcja selektora

Na zmiennych typu zbiorowego określona jest funkcja selektora **!** . Funkcja ta udostępnia jeden element zbioru usuwając go jednocześnie z tego zbioru. Kolejność udostępniania elementów zbioru nie jest określona.

Przykład 10.

```
...
type i_set : set of integer ;
var I : i_set ; i : integer ;
begin
   I := [ ] ;
   for i := 1 to 10 do I := I + [i] ;
   while I < > [ ] do write(!(I))
end ;
...
```

Wykonanie powyższego fragmentu programu spowoduje wypisanie liczb 1,..., 10 w pewnym (nie znanym z góry) porządku i usunięcie tych liczb ze zbioru I.

6.2.6. Wskaźnik zakończenia obliczeń

Globalna zmienna **all** o wartościach binarnych jest wskaźnikiem zakończenia obliczeń. Przyjmuje ona wartość true jeżeli wszystkie aktywne procesy są w stanie zawieszenia i wartość false w przeciwnym przypadku.

6.3. Program przykładowy

Program *binopt* znajduje za pomocą metody przeglądu sterowanego maksymalną wartość liniowej funkcji celu na zbiorze wektorów binarnych o długości n zadanych w sposób niejawny za pomocą zbioru liniowych ograniczeń. Formalny zapis problemu ma postać zadania optymalizacyjnego:

$$\max c^T x$$

przy ograniczeniach $Ax \leq b$ and $x \geq 0$, x - wektor binarny, gdzie A, b, c są macierzami i wektorami odpowiednich rozmiarów.

Zasadniczą częścią programu jest pętla główna (*main loop*), gdzie podproblemy (to jest problemy otrzymane z problemu orginalnego przez ustalenie wartości nie więcej niż $n-1$ współrzędnych wektora x) są badane (procedura *fathom*) a znalezione rozwiązania (wektory w których ustalono wartość wszystkich współrzędnych) są oceniane.

Podproblem jest zamykany i usuwany z dalszych rozważań jeżeli na podstawie odpowiednich testów można stwierdzić, że ze wzgledu na zmienne których wartość została ustalona, niezależnie od wartości przyjmowanych przez pozostałe zmienne, rozwiązanie podproblemu może być jedynie albo niedpouszczalne (ze względu na organiczenia) albo też wartość funkcji dla tego rozwiązania będzie mniejsza bądź równa wartości uzyskanej dla innego znalezionego już rozwiązania i przechowywanej w zmiennej *incumbent*.

Jeżeli podproblem nie zostanie zamknięty, to zostaje on podzielony na dwa podproblemy poprzez wybór współrzędnej, której wartość dotychczas nie została ustalona i przez ustalenie jej wartości na 0 (pierwszy podproblem) i 1 (drugi podproblem). Postępowanie to prowadzi do utworzenia nowych podproblemów (w których pewne współrzędne nie mają jeszcze ustalonych wartości) lub do wyge-

nerowania rozwiązania (wszystkie współrzędne mają wartości ustalone).

Nowe podproblemy dołączane są do listy podproblemów. Rozwiązania są testowane na dopuszczalność i jeżeli są dopuszczalne obliczana jest dla nich wartość funkcji (procedura *value*). Jeżeli wartość funkcji jest większa niż dla najlepszego rozwiązania, to rozwiązanie podproblemu zostaje zapamiętane jako rozwiązanie aktualnie najlepsze.

Program kończy obliczenia jeżeli lista aktywnych podproblemów jest pusta. Wtedy rozwiązanie aktualnie najlepsze jest rozwiązaniem najlepszym dla całego problemu. Na początku obliczeń rozwiązanie aktualnie najlepsze jest nieokreślone natomiast przyjmuje się, że wartością rozwiązania aktualnie najlepszego jest $-\infty$. Dostęp do zmiennej z , która przechowuje wartość rozwiązania aktualnie najlepszego, jest kontrolowanyza pomocą zmiennej *semaphor.*

```
program binopt(input,output) ;
const n=1000 ; m=1000 ;
type  bin = 0..1 ;
      t = array[1..(n + 1)]of bin ;
      x = array[1..n] of bin ;
      y = set of x ;
var
   z           : real ;
   semaphor    : bin ;
   a           : array [1..m,1..n] of real ;
   b           : array [1..m] of real ;
   c           : array [1..n] of real ;
   incumbent   : x ;
   subproblem  : t ;
   s           : y ;
co_routine subproblem(subproblem : t ;) ;
var
   w     : real ;
   e,f   : Boolean ;
```

```
   i      : integer ;
begin
if subproblem(n+1) < > n then
   begin
   fathom(subproblem,e) ;
   if e = false then
      begin
      subproblem(n+1):=subproblem(n+1)+1 ;
      subproblem(subproblem(n+1)):=0 ;
      s:=s+[subproblem] ;
      subproblem(subproblem(n+1)):=1 ;
      s:=s + [subproblem]
      end
   end
   else
   begin
   feasible(subproblem,e) ;
   if e = true then
      begin
      value(subproblem,w) ;
      while semaphore < > 0 do ;
      semaphore := 1 ;
      if w z then
         begin
         z := w ;
         for i := 1 to n do
            incumbent(i) := subproblem(i)
         end ;
      semaphore:=0
      end
   end
end
procedure fathom(subproblem : t ; e : Boolean) ;
...
procedure feasible(subproblem : t ; e : Boolean) ;
...
procedure value(subproblem : t ; w : integer) ;
...
begin
{ Main loop }
```

```
for i := 1 to n + 1 do
   subproblem(i) := 0 ;
s := subproblem ;
until not all do
   for subproblem in s create parallel
      subproblem(subproblem)
end.
```

6.4. Uwagi końcowe

Wielość istniejących języków programowania wynika z różnorodności zadań rozwiązywanych za pomocą maszyn cyfrowych. W zasadzie każde zagadnienie da się rozwiązać w każdym języku programowania ale ilość pracy włożonej w to rozwiązanie oraz efektywność obliczeń silnie zależą od wybranego języka. Na przykład, do rozwiązywania zagadnień polegających na rekurencyjnym przeglądaniu struktur typu drzewo język Pascal jest wygodniejszy od języka Fortran. Z kolei w przypadku zagadnień z zakresu algebry liniowej Fortan jest wygodniejszy, a programy w nim napisane działają na ogół szybciej.

Z chwilą wejścia do eksploatacji maszyn wieloprocesorowych pojawił się jeszcze dodatkowy czynnik w postaci różnorodności architektur. Zupełnie odmienne konsekwencje dla procesu tworzenia oprogramowania mają na przykład architektury typu pocesor wektorowy i transputer. W pierwszym przypadku użyjemy raczej języka wyposażonego w operacje wektorowe (takie jak w języku Parallel Pascal), w drugim języka dającego możliwość elastycznego wykorzystywania sieci procesorów (np. Ocean).

Niewątpliwie język Modest jest bliższy zastosowaniom drugiego typu. Jakkolwiek język ten może być uznany za język ogólnego zastosowania, jego konstrukcja motywowana była głównie problemami obliczeń numerycznych ze szczególnym uwzględnieniem problemów kombinatorycznych. Stąd dość specyficzne potraktowanie

wymiany informacji i rozróżnienie programowania współbieżnego i równoległego.

Dynamiczne i rekurencyjne tworzenie procesów daje programiście możność elastycznego doboru strategii rozwiązywania zagadnień kombinatorycznych z uwzględnieniem nieograniczonej bądź ograniczonej liczby procesów. Jest to o tyle ważne, że zwykle trudno jest ustalić, lub jest to wręcz niemożliwe do ustalenia z góry, jaka strategia jest najefektywniejsza. Posługując się językiem Modest można łatwo zaprogramować rozmaite strategie mieszane, których dobór uzależniony byłby stanem obliczeń.

Niniejszy rozdział oparty jest w całości na pracy [2] napisanej w języku angielskim.

Literatura

1. Iglewski M., Madey J., Matwin S. (1978): Pascal. WNT, Warszawa.
2. Kaliszewski I., Wojtowicz M. (1991): Modest: A language for parallel programming. *Prace IPI PAN*, 697, Warszawa.
3. Reeves A.P. (1984): "Parallel Pascal: An Extended Pascal for Parallel Computers". *J. Parallel and Distributed Computing*, vol 1, 64 - 80.
4. Wirth N. (1972): "The Programming Language Pascal (Revised Report)". *Bericht der Fachgruppe Computer Wissenschaften*, Eidgenossische Technische Hochschule, Zurich.

Literatura dodatkowa

1. Burns A. (1988): Programming in Occam 2. Addison-Wesley Publ. Co.

2. Gutkowski R., Szturmowicz M. (1989): "Architektura transputera i język OCCAM". *Prace Instytutu Podstaw Informatyki PAN*, 651.
3. Hoare C.A.R. (1985): Communicating Sequential Processes. Prentice-Hall International, Englewood Cliffs.
4. Iszkowski W., Maniecki M. (1982): Programowanie współbieżne. WNT, Warszawa.
5. Sharp J. (1987): Introduction to distributed and parallel processing. Blackwell, Oxford.

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.