ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1994
Seria: AUTOMATYKA z.114 Nr kol. 1250

Jacek BŁAŻEWICZ, Maciej DROZDOWSKI, Jan WĘGLARZ
Politechnika Poznańska

SZEREGOWANIE ZADAŃ WIELOPROCESOROWYCH NA ALTERNATYWNYCH ZBIORACH PROCESORÓW DEDYKOWANYCH*

Streszczenie: W pracy rozważany jest problem szeregowania zadań, z których każde może być wykonywane w każdej chwili przez zbiór procesorów z pewnej skończonej rodziny zbiorów. Dla przypadku zadań niepodzielnych i dwóch lub trzech procesorów podane zostaną algorytmy dokładne wykorzystujące programowanie dynamiczne. Dla dowolnej liczby procesorów podany zostanie algorytm przybliżony. Gdy zadania są podzielne, a liczba procesorów ustalona, problem można rozwiązać w sposób dokładny wykorzystując programowanie liniowe.

SCHEDULING MULTIPROCESSOR TASKS ON ALTERNATIVE SETS OF PROCESSORS

Summary: In this paper we consider the problem of scheduling tasks requiring alternative sets of processors. In the case of nonpreemptive schedules and two or three processors exact algorithms based on dynamic programming are constructed. When the number of processors is not fixed an approximate algorithm is proposed. In the case of preemptive schedules and the fixed number of processors the problem can be solved using linear programming.

EINORDNUNG VON MULTIPROCESSORAUFGABEN MIT BESCHRÄNKTEN MENGEN DEDIZIERTEN PROZESSOREN

Zusammenfassung: Im Beitrag wird ein Ablaufplan von Verrichtungen, die in jedem Moment alternative Prozessormengen brauchen, betrachtet. Für den Fall nicht präemptiven Verrichtungen und zwei oder drei Prozessoren werden exakte Algorithmen, die Dynamische Programmierrung ausnutzen, vorgeschlagen. Für den Fall einer beliebieger Prozessoranzahl wird ein approximativ Algorithm angeboten. Wenn die Verrichtungen präemptiv und Prozessoranzahl konstant sind, das Problem darf sich auf der Basis der Linearprogrammierrung exakt zu lösen.

* Badania zostały częściowo sfinansowane z Badań Własnych.

1. Wstęp

W klasycznej teorii deterministycznego szeregowania zadań ([2,4,8,10]) zakłada się, że każde zadanie żąda do swego wykonania w danej chwili tylko jednego procesora. Wobec szybkiego rozwoju technologii równoległych systemów komputerowych założenie to straciło na aktualności. Nietrudno wskazać na problemy, które można rozwiązywać równolegle na kilku komputerach, wykorzystując ich naturalny paralelizm. Ze względów efektywnościowych wskazane jest, aby pewne moduły programu były wykonywane równocześnie w czasie rzeczywistym [12]. Z drugiej strony, istnieje pewna grupa problemów związanych na przykład z przesyłaniem danych [13,15], testowaniem elementów elektronicznych [14], równoległymi systemami plików [1], gdzie równoczesne wykorzystanie wielu procesorów jest wymogiem technologicznym. Zadania żądające do swego wykonania więcej niż jednego procesora jednocześnie nazywać będziemy *zadaniami wieloprocesorowymi*.

Problem szeregowania zadań wieloprocesorowych był analizowany dla dwóch modeli. W pierwszym zakładano, że istotna jest *liczba* jednocześnie żądanych procesorów [6,7,9,11]. Zadanie mogło być wykonywane na dowolnych procesorach, pod warunkiem że otrzymało żądaną ich liczbę. W drugiem modelu istotna jest nie liczba, ale *zbiór* jednocześnie żądanych procesorów [5,14,15,16].

W tej pracy rozważany będzie problem deterministycznego szeregowania zadań wieloprocesorowych dla drugiego modelu. Zakładać będziemy mianowicie, że zadanie może być wykonywane przez pewną ograniczoną liczbę alternatywnych zbiorów procesorów dedykowanych. Przyjmiemy zatem, że dany jest zbiór zadań $T=\{T_1,...,T_n\}$ oraz zbiór procesorów P składający się z m dedykowanych procesorów $P_1,P_2,...,P_m$. Dla każdego zadania T_j określona jest rodzina zbiorów $S_j=\{D_1,...,D_{j\mu}\}$, na których zadanie T_j może być wykonywane. Każdy taki zbiór procesorów nazywać będziemy *konfiguracją* dla danego zadania. Dla każdej konfiguracji D_l, zadania T_j, określony jest czas wykonywania t_j^{Dl}. W przypadku zadań podzielnych zakładamy, że czasy wykonania zadania w różnych konfiguracjach można sumować jako wartości względne (innymi słowy, można sumować części w procentach do 100). Kryterium jakości uszeregowania jest jego długość (C_{max}).

Można wykazać [5,15], że zarówno dla uszeregowań niepodzielnych, jak i podzielnych powyższy problem jest w ogólności silnie NP-trudny. Dla wybranych przypadków szczególnych podane zostaną jednak algorytmy pseudowielomianowe. Następnie przedstawiony zostanie algorytm aproksymacyjny rozwiązujący problem w postaci ogólnej, a jego efektywność zostanie oszacowana dla najgorszego przypadku. Dla uszeregowań podzielnych optymalne uszeregowanie może być wyznaczone w wielomianowym czasie za pomocą algorytmu wykorzystującego programowanie liniowe.

2. Uszeregowania niepodzielne

Problem szeregowania zadań niepodzielnych żądających do swego wykonania jednocześnie zbioru procesorów jest silnie NP-trudny już dla trzech procesorów i $|S_j|=1$ [5], a NP-trudny dla dwóch procesorów. Poniżej podany zostanie algorytm pseudowielomianowy dla szeregowania na dwóch procesorach i dla szczególnego przypadku szeregowania na trzech procesorach.

2.1. Dwa procesory

Metoda wykorzystana w poniższym algorytmie opiera się na obliczaniu funkcji sprawdzającej istnienie dopuszczalnego uszeregowania o długości i dla zbioru zadań $T_1,...,T_j$. Funkcję tę można zdefiniować następująco:

$$f(i,j,x,y)=\begin{cases} 1 \Leftrightarrow \text{zbiór zadań } \{T_1,...,T_j\} \text{ można wykonać w dopuszczalny sposób w } i \\ \quad \text{jednostkach czasu, a procesory } P_1 \text{ i } P_2 \text{ pracują odpowiednio } x \text{ oraz } y \\ \quad \text{jednostek czasu} \\ 0 \quad \text{w przeciwnym razie} \end{cases}$$

Taką funkcję można obliczyć tylko wtedy, gdy czasy wykonania zadań są wyrażone za pomocą pewnej wspólnej jednostki (tj. są liczbami naturalnymi). Uszeregowanie optymalne ma długość równą najmniejszemu i, takiemu że $f(i,n,x,y)=1$ dla pewnych x, y. Zauważmy ponadto, że gdyby wykonywać wszystkie zadania po kolei, w konfiguracji o najkrótszym czasie wykonania, to uzyskane uszeregowanie byłoby dopuszczalne. Tak więc górnym ograniczeniem długości optymalnego uszeregowania jest suma czasów wykonania zadań w najkrótszej konfiguracji.

W celu wyprowadzenia wzorów określających poszukiwaną funkcję należy rozważyć, w jaki sposób każde z zadań może być uszeregowane. Poniżej rozważamy wszystkie możliwości.

Dla pierwszego zadania dopuszczalne uszeregowanie częściowe musi mieć długość równą czasowi wykonania tego zadania w jednej z dopuszczalnych konfiguracji. Załóżmy teraz, że zbadano już wszystkie uszeregowania częściowe dotyczące zadań $T_1,...,T_{j-1}$ dla $j \geq 2$. Dla zadania T_j mogą zajść następujące przypadki.

1) Zadanie zostaje uszeregowane w konfiguracji dwuprocesorowej. Wtedy bez utraty ogólności możemy przyjąć, że zadanie zostało umieszczone na początku uszeregowania.

2) Zadanie zostaje uszeregowane w konfiguracji jednoprocesorowej, na jednym z dwóch procesorów. Bez utraty ogólności przyjmiemy, że w uszeregowaniu częściowym zadanie jest wykonywane jako ostatnie na danym procesorze. Należy tu wyróżnić trzy podprzypadki:

2α) Długość uszeregowania nie zmienia się wskutek uszeregowania zadania T_j (tj. długość uszeregowania jest równa długości uszeregowania dla zadań $T_1,...,T_{j-1}$).

2β) Wskutek uszeregowania T_j uszeregowanie wydłuża się, ale procesor, na którym umieszczono T_j, nie był najbardziej obciążonym procesorem dla zadań $T_1,...,T_{j-1}$.

2γ) Wskutek uszeregowania T_j uszeregowanie wydłuża się, a procesor, na którym umieszczono T_j, był najbardziej obciążonym procesorem dla zadań $T_1,...,T_{j-1}$.

Stosownie do opisanych powyżej sytuacji funkcja $f(i,j,x,y)$ przyjmuje wartość 1 w wymienionych poniżej przypadkach:

dla $j=1$:

$f(i,1,i,i)=1$ gdy $t_1^{12}=i$ (T_1 zostaje uszeregowane na P_1,P_2),
$f(i,1,i,0)=1$ gdy $t_1^{1}=i$ (T_1 zostaje uszeregowane na P_1),
$f(i,1,0,i)=1$ gdy $t_1^{2}=i$ (T_1 zostaje uszeregowane na P_2),

dla $j>1$:

$f(i-t^{12},j-1,x-t^{12},y-t^{12})=1$ $x,y\le i$ (T_j zostaje uszeregowane na P_1P_2 - przypadek 1),
$f(i,j-1,x-t^{1},i)=1$ $x\le i$ (T_j zostaje uszeregowane na P_1 - przypadek 2α),
$f(y,j-1,i-t^{1},y)=1$ $i-t_j^{1}\le y\le i$ (T_j zostaje uszeregowane na P_1 - przypadek 2β),
$f(i-t^{1},j-1,i-t^{1},y)=1$ $y\le i-t_j^{1}$ (T_j zostaje uszeregowane na P_1 - przypadek 2γ),
$f(i,j-1,i,y-t^{2})=1$ $y\le i$ (T_j zostaje uszeregowane na P_2 - przypadek 2α),
$f(x,j-1,x,i-t^{2})=1$ $i-t_j^{2}\le x\le i$ (T_j zostaje uszeregowane na P_2 - przypadek 2β),
$f(i-t^{2},j-1,x,i-t^{2})=1$ $x\le i-t_j^{2}$ (T_j zostaje uszeregowane na P_2 - przypadek 2γ),

we wszystkich pozostałych przypadkach $f(i,j,x,y)=0$,

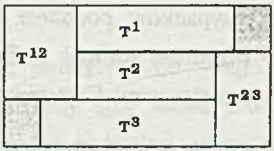
$$i=1,...\sum_{l=1}^{n}\min_{D_p\in S_l}\{t_l^{D_p}\},\ j=2,...n,\ x,y=0,...,\sum_{l=1}^{n}\min_{D_p\in S_l}\{t_l^{D_p}\}.$$

Optymalne uszeregowanie ma długość $C^*_{\max}=\min_{x,y}\{i: f(i,n,x,y)=1\}$ i można je znaleźć w sposób typowy dla programowania dynamicznego, przeszukując tablicę funkcji $f(i,j,x,y)$ począwszy od $f(C^*_{max},n,x,y)=1$, a skończywszy na takim elemencie, że $j=1$ i funkcja ma wartość 1. W pojedynczym kroku przeszukiwania tablicy funkcji $f(i,j,x,y)$ badamy, dla jakich i',x',y' zachodzi $f(i',j-1,x',y')=1$, co przy założeniu jednego ze sposobów uszeregowania T_j umożliwiło zajście $f(i,j,x,y)=1$. Określamy tym samym, w jaki sposób zadanie zostało uszeregowane (zależnie od tego, który z powyższych siedmiu przypadków zachodzi).

Przedstawiony powyżej algorytm ma złożoność $O(n(\sum_{l=1}^{n}\min_{D_p\in S_l}\{t_l^{D_p}\})^3)$, co wynika z konieczności obliczenia wartości funkcji $f(i,j,x,y)$, w najgorszym przypadku, dla wszystkich wartości i, j, x, y. Ponieważ funkcja złożoności obliczeniowej zależy od stałych liczbowych występujących w problemie, jest to algorytm pseudowielomianowy.

2.2. Trzy procesory

Jak wspomnieliśmy, problem szeregowania zadań niepodzielnych, żądających do swego wykonania zbioru procesorów jest silnie NP-zupełny już dla trzech procesorów. Jeżeli jednak w zbiorze dopuszczalnych konfiguracji nie występuje jedna z konfiguracji dwuprocesorowych, to możliwe jest podanie algorytmu pseudowielomianowego, podobnego do przedstawionego w poprzednim rozdziale. Bez utraty ogólności przyjmijmy, że zadania nie wymagają jednocześnie do swego wykonania procesorów P_1, P_2. Każde uszeregowanie dopuszczalne można wówczas przekształcić do postaci z rys. 1 nie zwiększając jego długości.



Rys. 1. Standardowa postać uszeregowania zadań niepodzielnych na trzech procesorach.
Fig. 1. Standard form of a schedule for nonpreemptive tasks and three processors.

W celu wyznaczenia uszeregowania optymalnego posłużymy się metodą podobną do przedstawionej w poprzednim rozdziale i będziemy obliczać funkcję $f(i,j,x,y,z)$ sprawdzającą istnienie uszeregowania o długości i dla zadań $T_1,...,T_j$. Ściślej, funkcję tę można zdefiniować następująco:

$$f(i,j,x,y,z) = \begin{cases} 1 \Leftrightarrow \text{zbiór zadań } \{T_1,...,T_j\} \text{ można wykonać w dopuszczalny sposób w } i \\ \quad \text{jednostkach czasu, a procesory } P_1,P_2,P_3 \text{ pracują odpowiednio } x,y \\ \quad \text{oraz } z \text{ jednostek czasu} \\ 0 \quad \text{w przeciwnym razie} \end{cases}$$

Nim podamy wzory pozwalające wyznaczyć $f(i,j,x,y,z)$, rozważmy kiedy funkcja ta może przyjąć wartość 1. Każde z zadań T_j, $j=1,...,n$ może zostać uszeregowane w jeden z następujących sposobów:

1) Zadanie T_j zostaje uszeregowane na wszystkich trzech procesorach. Bez utraty ogólności można przyjąć, że zadanie takie znajduje się na początku uszeregowania (przed wszystkimi zadaniami dwu- i jednoprocesorowymi), wtedy także $x,y,z<i$, gdyż w uszeregowaniu o długości i żaden z procesorów nie może pracować dłużej niż i.

2) Zadanie T_j zostaje uszeregowane w trybie dwuprocesorowym na procesorach P_1,P_2 albo P_2,P_3.

3) Zadanie T_j zostaje uszeregowane w trybie jednoprocesorowym na jednym z trzech procesorów.

W każdym z dwóch ostatnich przypadków należy wyróżnić trzy dalsze podprzypadki:

α) Zadanie T_j zostaje przydzielone w ten sposób, że długość całego uszeregowania jest wyznaczona przez te same procesory, które określały długość uszeregowania dla $T_1,...,T_{j-1}$.

β) Zadanie T_j zostaje przydzielone w taki sposób, że długość uszeregowania dla $T_1,...,T_j$ jest wyznaczana przez moment zakończenia T_j, dla $T_1,...,T_{j-1}$ najbardziej obciążonymi procesorami nie były procesory wykonujące T_j.

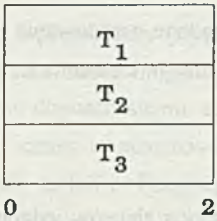
γ) T_j zostaje przydzielone w ten sposób, że długość uszeregowania dla $T_1,...,T_j$ jest wyznaczana przez moment zakończenia T_j, dla $T_1,...,T_{j-1}$ najbardziej obciążonymi procesorami były pewne procesory wykonujące T_j.

Dla wymienionych powyżej przypadków, podobnie jak w poprzednim rozdziale, można określić kiedy funkcja $f(i,j,x,y,z)$ przyjmuje wartość 1. Ze względu na ograniczoną objętość pracy wyprowadzenie wzorów pozostawiamy Czytelnikowi. Uszeregowanie optymalne ma długość $C^*_{max} = \min_{x,y,z}\{i: f(i,n,x,y,z) = 1\}$, a można je wyznaczyć w taki sam sposób jak dla przypadku dwóch procesorów, przeszukując tablicę funkcji f od $f(C^*_{max},n,x,y,z)=1$ aż do znalezienia elementu takiego, że $j=1$, a wartość funkcji jest 1. Złożoność powyższego pseudowielomianowego algorytmu wynika z konieczności wyliczenia, w najgorszym przypadku, wszystkich wartości funkcji f. Wymaga to $O(n(\sum_{l=1}^{n} \min_{D_p \in S_l}\{t_l^{D_p}\})^4)$ jednostek czasu. Rozpatrzmy prosty przykład.

PRZYKŁAD. Dany jest zbiór zadań $T=\{T_1,T_2,T_3\}$, oraz konfiguracje: $S_1=\{\{P_1\},\{P_1P_2\},\{P_1P_2P_3\}\}$, $S_1=\{\{P_2\},\{P_1P_2\},\{P_2P_3\}\}$, $S_1=\{\{P_1\},\{P_2\},\{P_3\}\}$. Czasy wykonywania zadań w poszczególnych konfiguracjach wynoszą: $t_1^1=2$, $t_1^{12}=1$, $t_1^{123}=2$, $t_2^2=2$, $t_2^{12}=2$, $t_2^{23}=3$, $t_3^1=1$, $t_3^2=2$, $t_3^3=2$. Górne ograniczenie na długość uszeregowania $\sum_{l=1}^{4} \min_{D_p \in S_l}\{t_l^{D_p}\}=4$. Niezerowe wartości funkcji f zostały przedstawione w tablicy 1, a uszeregowanie optymalne na rys. 2.

Tablica 1. Wartości funkcji $f(i,j,x,y,z)$.
Podano tylko wartości różne od 0

j \ i	1	2	3	4
1	(1,1,0)	(2,0,0) (2,2,2)		
2		(2,2,0)	(1,3,0) (3,3,0) (2,3,3)	(2,4,2) (2,3,0) (3,3,3) (2,4,3)
3		(2,2,2)	(3,2,0) (2,3,0) (3,3,3) (1,3,2) (3,3,2)	(4,3,0) (3,4,2) (2,4,3) (2,4,0) (2,4,4) (4,2,2) (4,4,4)



Rys. 2. Przykład. Uszeregowanie optymalne
Fig. 2. Example. The optimal schedule

2.3. Dowolna liczba procesorów

Problem szeregowania zadań, żądających do swego wykonania jednocześnie zbioru procesorów staje się jeszcze trudniejszy, gdy liczba procesorów jest dowolna.. Do jego rozwiązania można zastosować algorytm podziału i ograniczeń zaproponowany w [3]. Poniżej przedstawimy natomiast algorytm przybliżony, dla którego określona zostanie, dla najgorszego przypadku, odległość generowanych przez niego rozwiązań od optimum. Algorytm ten wykorzystuje konfigurację zadania, która ma najkrótszy czas wykonania.

Algorytm NK (*Najkrótsza Konfiguracja*): Wykonuj zadania w dowolnej kolejności w konfiguracji o najkrótszym czasie wykonania. W uszeregowaniu przesuń zadania maksymalnie w lewo.

Dla powyższego algorytmu można podać oszacowanie w najgorszym przypadku. Przyjmijmy, że $C_{\max}^{NK}$ to długość uszeregowania wygenerowanego przez algorytm NK.

Twierdzenie. *Dla algorytmu NK zachodzi* $\frac{C_{\max}^{NK}}{C_{\max}^{*}} \le m$ *i oszacowanie to jest osiągane.*

Dowód. Zauważmy, że dla dowolnego uszeregowania (także optymalnego) zachodzi: $C_{\max}^{*} \ge \frac{1}{m}\sum_{l=1}^{n} \min_{D_p \in S_l} \{t_l^{D_p}\}$, gdyż zadania można uszeregować tak, jak gdyby były podzielne, a każde zadanie wymagające l procesorów można potraktować jak l zadań jednoprocesorowych. Z drugiej strony, uszeregowanie zbudowane z zadań wykonywanych w najkrótszej konfiguracji, w dowolnej kolejności i bez dosunięcia zadań w lewo ma długość nie mniejszą niż $C_{\max}^{NK}$, czyli: $C_{\max}^{NK} \le \sum_{l=1}^{n} \min_{D_p \in S_l} \{t_l^{D_p}\}$. Uwzględniając powyższe dwie zależności otrzymujemy tezę twierdzenia.

Musimy jeszcze wykazać, że podane oszacowanie jest osiągane. W tym celu rozważmy następujący przykład: zbiór zadań ma n elementów, każde z zadań ma czasy wykonania określone następująco:

$t_j^1 = a$ dla $j=1,...,n$,

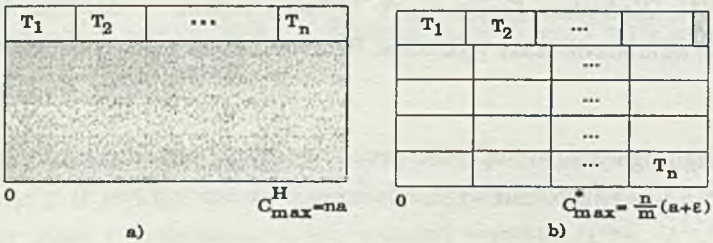
$t_j^k = a+\varepsilon$ dla j=1,...n oraz dla wszystkich zbiorów jednoelementowych $k=\{2\},...,\{m\}$, gdzie a, $\varepsilon < \frac{am}{n}$ są pewnymi stałymi.

W innych konfiguracjach zadania nie mogą być wykonywane. Uszeregowanie wytworzone przez algorytm NK ma długość $C_{max}^{NK} = na$ (rys.3a), natomiast uszeregowanie optymalne ma długość $C_{max}^{*} = \frac{n}{m}(a+\varepsilon)$ (rys.3b). Stąd,

$$\lim_{\varepsilon \to 0} \frac{C_{max}^{NK}}{C_{max}^{*}} = \lim_{\varepsilon \to 0} \frac{m}{1+\frac{\varepsilon}{a}} = m.$$

□

Powyższe oszacowanie wskazuje na potrzebę dalszych poszukiwań algorytmów o lepszych gwarancjach jakości.



Rys.3. Przykład sytuacji, w której algorytm NK osiąga oszacowanie m
Fig. 3. An example case when bound m for algorithm NK is attained

3. Uszeregowania podzielne

W przypadku zadań podzielnych problem staje się silnie NP-trudny już dla $|S_j|=1$ $(j=1,...,n)$, jeżeli liczba procesorów jest dowolna [15]. Jeśli natomiast liczba procesorów jest ustalona, to problem można rozwiązać w czasie ograniczonym przez wielomian zależny od liczby zadań. Poniżej przedstawiamy algorytm rozwiązujący w czasie wielomianowym problem szeregowania podzielnych zadań wieloprocesorowych, żądających określonych zbiorów procesorów jednocześnie, przy założeniu ustalonej liczby procesorów. Algorytm ten wykorzystuje programowanie liniowe.

W celu przedstawienia algorytmu wprowadźmy pojęcie *dopuszczalnego zbioru zadań*. Dopuszczalny zbiór zadań tworzą zadania, które mogą być wykonywane równocześnie na danym zbiorze procesorów. Z każdym dopuszczalnym zbiorem zadań i możemy związać zmienną x_i, która oznaczać będzie, ile czasu w poszukiwanym uszeregowaniu optymalnym wykonywany będzie dopuszczalny zbiór zadań i. Przyjmijmy, że **G** to rodzina wszystkich dopuszczalnych zbiorów zadań, a $G_j^{D_l}$ to zbiór numerów dopuszczalnych zbiorów zadań, w których zadanie T_j jest wykonywane w konfiguracji wykorzystującej zbiór procesorów D_l. Możemy teraz sformułować nasz problem jako zadanie programowania liniowego.

zminimalizować $\sum_{i \in G} x_i$

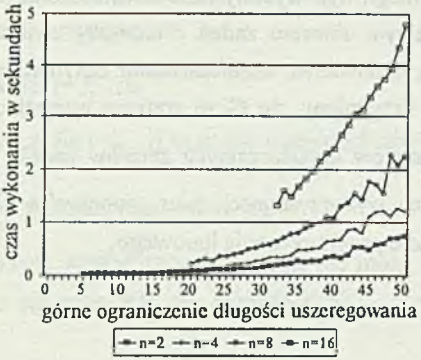
przy ograniczeniach $\sum_{D_l \in S_j} \sum_{i \in G_j^{D_l}} \frac{x_i}{t_j^{D_l}} \geq 1$ dla $j=1,...,n$

Powyższe sformułowanie zawiera n ograniczeń i $O(n^m)$ zmiennych. Gdy m jest ustalone, liczba zmiennych jest ograniczona przez wielomian od liczby zadań. Ponieważ zadanie programowania liniowego można rozwiązać w czasie ograniczonym przez wielomian od liczby zmiennych i ograniczeń, nasz problem jest obliczeniowo łatwy dla ustalonego m.

4. Ocena symulacyjna

W tym rozdziale przedstawiamy wyniki eksperymentów obliczeniowych przeprowadzonych nad przedstawionymi wcześniej algorytmami. Zaprezentowane zostaną zależności czasu wykonania i zajętości pamięci od rozmiarów instancji.

Program symulacyjny został napisany dla komputera IBM AT-386 w języku Pascal, wykorzystano system operacyjny MSDOS. Parametry opisujące zadania były generowane pseudolosowo w następujący sposób: w pierwszej kolejności z rozkładu równomiernego generowana była liczba konfiguracji, w których zadanie może być wykonywane (dla szeregowania na dwóch procesorach liczba ta wynosi 3, a dla uszeregowań podzielnych przyjęto maksymalnie 6 konfiguracji na zadanie). Następnie, również z rozkładu równomiernego, generowane były zbiory procesorów żądane w poszczególnych konfiguracjach oraz czasy wykonania zadań dla poszczególnych konfiguracji. Zależności czasu wykonania algorytmów od rozmiarów instancji przedstawiono na rys.4 i rys.7, a zajętości pamięci na rys.5 i rys.8. Na rys. 6 pokazano zależność średniej odległości od optimum i odległości w najgorszym zaobserwowanym przypadku dla algorytmu **NK**. Na rys. 9 przedstawiono zależność liczby zmiennych w zadaniu programowania liniowego (dla algorytmu szeregowania zadań podzielnych) od liczby zadań.



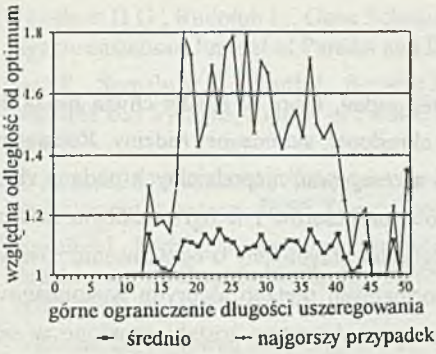
Rys. 4. Zależność czasu wykonania algorytmu szeregowania zadań niepodzielnych na dwóch procesorach od rozmiaru instancji

Fig. 4. Execution time of the algorithm scheduling nonpreemptive tasks on two processors vs. the size of the instance

Rys. 5. Zależność zajętości pamięci dla algorytmu szeregowania zadań niepodzielnych na dwóch procesorach od rozmiaru instancji

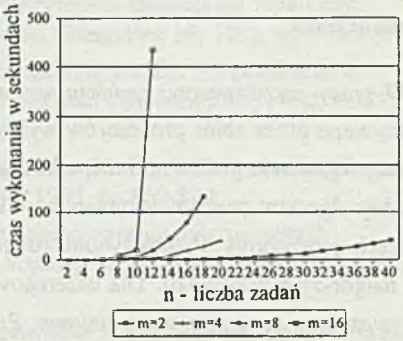
Fig. 5. Memory consumption of the algorithm scheduling nonpreemptive tasks on two processors vs. the size of the instance

Na rys. 4 można zauważyć, że czas wykonania algorytmu dla dwóch procesorów rośnie szybko wraz z górnym ograniczeniem długości uszeregowania (ale wolniej niż wykładniczo). W opisywanej implementacji udało się uzyskać niewielką zajętość pamięci (rys.5) dzięki przechowywaniu jedynie wartości funkcji $f(i,j,x,y)$ różnych od 0. We wszystkich przeprowadzonych eksperymentach czas wykonania heurystyki NK był krótszy niż 60ms, natomiast jakość średnio odbiegała od optimum o 10-20% (rys. 6). Z drugiej strony obserwowano także przykłady potwierdzające oszacowanie w najgorszym przypadku. Zależność czasu wykonania algorytmu (rys.7) i zajętości pamięci (rys.8), dla algorytmu szeregowania zadań podzielnych, od liczby zadań i procesorów pokazuje, że parametry te rosną szybko z liczbą procesorów, natomiast znacznie wolniej - z liczbą zadań. Porównanie z rys. 9 pokazuje, że rozwiązywane zadania programowania liniowego nie były bardzo duże, a więc najistotniejszą część całego czasu wykonania algorytmu zajmuje wyznaczenie dopuszczalnych zbiorów zadań.



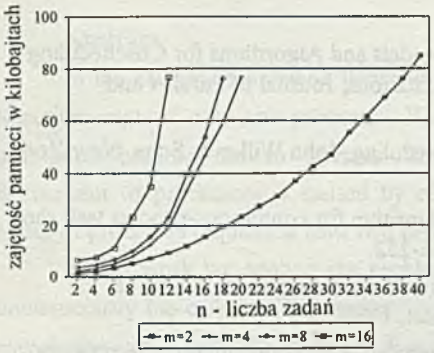
Rys. 6. Zależność odległości od optimum uszeregowania, generowanego przez algorytm NK, od rozmiaru instancji (n=4)

Fig. 6. Distance from the optimum of the solution generated by NK algorithm vs. the size of the instance (n=4).



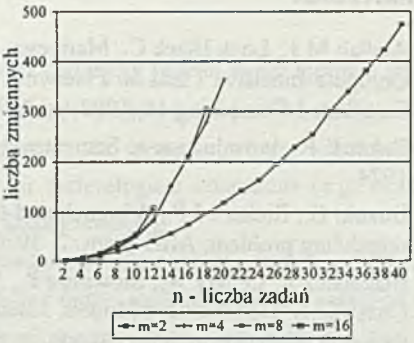
Rys. 7. Zależność czasu wykonania algorytmu szeregowania zadań podzielnych od rozmiaru instancji

Fig. 7. Execution time of the algorithm for preemptive tasks vs. the size of the instance



Rys.8. Zależność zajętości pamięci dla algorytmu szeregowania zadań podzielnych od rozmiaru instancji

Fig. 8. Memory consumption of the algorithm for preemptive tasks vs. the size of the instance



Rys.9. Liczba zmiennych w zadaniu programowania liniowego dla algorytmu szeregowania zadań podzielnych od rozmiaru instancji

Fig. 9. Number of variables in the linear program solving the preemptive scheduling problem vs. the size of instance

Podsumowując wyniki uzyskane w serii eksperymentów obliczeniowych, możemy stwierdzić, że algorytmy przedstawione we wcześniejszych rozdziałach zachowują się w sposób zadowalający zarówno pod względem szybkości, jak i wymagań pamięciowych.

5. Zakończenie

W pracy przedstawiono problem szeregowania zadań, które w każdej chwili mogą być wykonywane przez zbiór procesorów wybrany z określonej, skończonej rodziny. Rozważane były uszeregowania podzielne i niepodzielne. Dla uszeregowań niepodzielnych podany został dokładny algorytm pseudowielomianowy dla dwóch procesorów i w ograniczonym zakresie dla trzech procesorów. Przedstawiono również algorytm przybliżony o oszacowaniu równym m w najgorszym przypadku. Dla uszeregowań podzielnych podano algorytm wielomianowy wykorzystujący programowanie liniowe. Przedstawione algorytmy zostały ocenione w serii eksperymentów obliczeniowych.

Dalsze poszukiwania mogą dotyczyć algorytmów przybliżonych o lepszych oszacowaniach lub uwzględnienia innych kryteriów optymalności np. kryterium L_{max}, istotnego zwłaszcza w środowisku czasu rzeczywistego.

LITERATURA

[1] Atallah M.J., Lock Black C., Marinescu D.C., Models and Algorithms for Coscheduling Compute-Intensive Tasks on a Network of Workstations, Journal of Parallel and Distributed Computing 16, 1992, pp.319-327.

[2] Baker K.R., Introduction to Sequencing and Scheduling, John Willey & Sons, New York, 1974.

[3] Bozoki G., Richard J.P., A branch-and-bound algorithm for continuous-process task shop scheduling problem, AIIE Trans. 2, 1970, pp.246-252.

[4] Błażewicz J., Cellary W., Słowiński R., Węglarz J., Scheduling under Resource Constraints: Deterministic Models, J.Baltzer, Basel, 1986.

[5] Błażewicz J., Dell'Olmo P., Drozdowski M., Speranza M.G, Scheduling multiprocessor tasks on three dedicated processors, IPL 41, 1992, pp.275-280.

[6] Błażewicz J., Drabowski M., Węglarz J., Scheduling multiprocesor tasks to minimize schedule length, IEEE Trans. Comput. 35, 1986, pp.389-393.

[7] Błażewicz J., Drozdowski M., Schmidt G., de Werra D., Scheduling independent multiprocessot tasks on a uniform k-processor system, Parallel Computing, 1994, przyjęte do druku.

[8] Błażewicz J., Ecker K., Schmidt G., Węglarz J., Scheduling in Computer and Manufacturing Systems, Springer, New York, 1993.

[9] Chen G.I., Lai T.H., Preemptive scheduling of independent jobs on a hypercube, IPL 28, 1988, pp.201-206.

[10] Coffman Jr. E.G.(ed.), Computer and Job-Shop Scheduling Theory, John Willey & Sons, New York, 1976.

[11] Du J., Leung J.Y-T., Complexity of scheduling parallel task systems, SIAM J.Discrete math. 2, 1989, pp.473-487.

[12] Feitelson D.G., Rudolph L., Gang Scheduling Performance Benefits for Fine-Grain Synchronization, Journal of Parallel and Distributed Computing 16, 1992, pp.306-318.

[13] Jain R., Somalwar K., Werth J., Browne J.C., Scheduling parallel I/O operations in Multiple Bus Systems, Journal of Parallel and Distributed Computing 16, 1992, pp.352-262.

[14] Krawczyk H., Kubale M., An approximation algorithm for diagnostic test scheduling in multicomputer systems, IEEE Trans. Comput. 34, 1985, pp.869-872.

[15] Kubale M., Podzielne uszeregowania zadań dwuprocesorowych na procesorach dedykowanych, Zeszyty Naukowe Politechniki Śląskiej, Seria: Automatyka z.100, Nr kol. 1082, 1990, pp.147-153.

[16] Veltman B., Lageweg B.J., Lenstra J.K., Multiprocessor scheduling with communication delays, Parallel Computing 16, 1990, pp.173-182.

Recenzent: Prof. dr hab. inż Jerzy Klamka

Wpłynęło do Redakcji do 30.04.1994 r.

Abstract

In the classical scheduling theory it is assumed that every task in every moment of its execution requires only one processor. With the rapid development of parallel systems this assumption became not so obvious. There are computer systems where simultaneous requirement of processors is caused by efficiency or technological conditions (e.g. mutual testing of procesors requires at least two processors simulatneously).

In this work we analyse the problem of scheduling tasks requirng sets of processors simultaneously (so-called multiprocessor tasks), what is more, each task can be executed by several alternative sets of processors. For the case of nonpreemptive schedules and two or three processors exact algorithms based on a dynamic programming procedure has been proposed. When the number of processors is not fixed an approximate algorithm has been proposed. For the case of preemptive schedules and the fixed number of processors the problem can be solved by an exact algorithm based on linear programming.