Adam PRUS, Krzysztof PIEŃKOSZ
Politechnika Warszawska

SZEREGOWANIE ZADAŃ CZĘŚCIOWO PODZIELNYCH NA PROCESORACH RÓWNOLEGŁYCH

Streszczenie. W pracy jest rozpatrywany problem szeregowania zadań na identycznych procesorach równoległych. Celem jest uzyskanie harmonogramu o minimalnej długości. Zakłada się, że zadania są niezależne i częściowo podzielne tzn. mogą być dzielone na fragmenty wykonywane na różnych procesorach, ale każdy z tych fragmentów nie może być krótszy od zdanego parametru k. Przedstawiono heurystyczne algorytmy szeregowania zadań częściowo podzielnych dla przypadku, gdy występują tylko dwa procesory oraz dla przypadku ogólnego z dowolną liczbą procesorów.

TASK SCHEDULING WITH RESTRICTED PREEMPTIONS ON PARALLEL IDENTICAL PROCESSORS

Summary. In the paper the task scheduling problem on parallel identical processors is considered. The objective is to find the schedule of minimum makespan. We assume that tasks are independent and can be preempted but preemption is only allowed if the task is continuously processed for at least k units of time. We propose heuristic scheduling algorithms for the case with only two processors and for general case with arbitrary number of processors.

1. Wprowadzenie

Klasyczne modele szeregowania zadań na procesorach równoległych, które są analizowane w literaturze (np. w [2, 6]), mają zwykle charakter dyskretny albo ciągły. W dyskretnych modelach szeregowania zakłada się, że zadania są niepodzielne, czyli że wykonywanie tych zadań nie może być przerywane i w związku z tym są one zawsze w całości wykonywane na jednym, wybranym procesorze. W modelach ciągłych, natomiast, dopuszczalne jest przerywanie zadań w dowolnym momencie i kończenie ich później na innych procesorach. Wykorzystując możliwość przerywania zadań możemy zazwyczaj uzyskać krótsze harmonogramy, jednakże przerywanie i przenoszenie zadań na inne procesory w celu dokończenia, wiąże się z różnego rodzaju niedogodnościami i dodatkowymi kosztami. Jeżeli tych przerwań jest dużo, to koszty te mogą przewyższać zyski wynikające z uzyskania krótszego harmonogramu.

Z tego powodu w rozpatrywanym w pracy semi-ciągłym modelu szeregowania zakłada się, że zadania są tylko częściowo podzielne, tzn. mogą być przerywane, ale

pod warunkiem, że każdy fragment tego zadania jest odpowiednio długo wykonywany na poszczególnych procesorach. Pozwala to na uzyskanie w elastyczny sposób kompromisu pomiędzy długością uszeregowania a kosztami przerwań.

Szczegółowy opis tego modelu i jego właściwości są zamieszczone w rozdziale 2. Następnie są przedstawione proponowane algorytmy szeregowania, przy czym rozdział 3 dotyczy szczególnego przypadku z tylko dwoma procesorami, natomiast rozdział 4 ogólnego wieloprocesorowego przypadku. Przykładowe wyniki eksperymentów obliczeniowych i porównań, a także wynikające z nich wnioski są zamieszczone rozdziale 5.

2. Sformułowanie i właściwości problemu

Niech N oznacza zbiór n niezależnych zadań, natomiast M zbiór m identycznych, równoległych procesorów, na których te zadania mogą być wykonane. Ponadto przyjmijmy, że p_j oznacza czas niezbędny do wykonania każdego z zadań $j \in N$, a $p_{\max}$ jest najdłuższym z tych czasów, tzn. $p_{\max} = \max\{p_j: j \in N\}$. W rozpatrywanym problemie szeregowania zakłada się, że zadania ze zbioru N są *częściowo podzielne*, co oznacza, że mogą być dzielone na fragmenty wykonywanie po kolei na różnych procesorach, ale każdy z tych fragmentów nie może być krótszy od wartości zadanego parametru k. Poszczególne fragmenty zadań wymagają do obsługi tylko jednego procesora ze zbioru M. Wartość k jest parametrem rozpatrywanego problemu szeregowania. Zauważmy, że gdy czas obsługi p_j pewnego zadania $j \in N$ jest krótszy niż $2k$, to takie zadanie nie może być dzielone i musi być w całości (bez przerywania) wykonane na jednym z procesorów.

Celem jest wyznaczenie najkrótszego harmonogramu wykonania zadań ze zbioru N na procesorach ze zbioru M. Długość takiego (optymalnego) harmonogramu będziemy oznaczać symbolem $C_{\max}$, natomiast do oznaczenia sformułowanego powyżej problemu szeregowania zadań użyjemy notacji $P \mid \textit{restricted k-pmtn} \mid C_{\max}$.

Identyczny problem szeregowania był rozpatrywany w pracy [1]. Przedstawiono tam dwa algorytmy heurystyczne TSRP3 i TSRP4 oraz eksperymentalnie zbadano i porównano ich efektywność.

W pracy [5] był analizowany szczególny przypadek tego problemu, gdy występują tylko dwa procesory. Sformułowano tam pewne warunki, przy których optymalne rozwiązanie może być w sposób efektywny wyznaczone. Wykorzystując te właściwości, zaproponowano dwa heurystyczne algorytmy 2RS i 2DS, które, w przypadku gdy $m = 2$, wyznaczały dużo duże lepsze harmonogramy niż algorytmy TSRP3 i TSRP4.

Podobne, ale nieco inne modele szeregowania zadań były analizowane w pracy [3]. W pierwszym z rozpatrywanych tam modeli zakładano, że podział zadania może nastąpić tylko wtedy, gdy zadanie to lub jego fragment jest przynajmniej przez k jednostek czasu wykonywane na danym procesorze w sposób nieprzerywany. W odróżnieniu od problemu $P \mid \textit{restricted k-pmtn} \mid C_{\max}$ ostatni fragment podzielonego zadania może być jednak krótszy od k. W drugim z rozpatrywanych modeli dopuszczano podział zadań tylko w momentach, będących wielokrotnością k. W pracy [3] wykazano, że w ogólnym przypadku oba problemy są *NP*-trudne.

Jeśli chodzi złożoność obliczeniową rozpatrywanego w tej pracy problemu $P \mid restricted\ k\text{-}pmtn \mid C_{\max}$, to zależy ona w sposób istotny od wartości k. Jeżeli $k > p_{\max}/2$, to żadne zadanie nie może być podzielone i w rezultacie mamy do czynienia z problemem $P \mid\mid C_{\max}$, który jest *NP*-trudny (patrz np. [2], [6]). Jednym z najczęściej stosowanych prostych algorytmów szeregowania dla takiego problemu jest algorytm przybliżony LPT (Longest Processing Time). Zgodnie z nim zadania są umieszczane na wolnych procesorach w kolejności nierosnących czasów obsługi.

Z drugiej strony, jeżeli $k = 0$, to zadania są całkowicie podzielne i stosując regułę McNaughtona [4] można je optymalnie uszeregować w czasie $O(n)$. Optymalna długość uszeregowania jest wtedy równa

$$C^* = \max\left\{p_{max},\ \frac{\sum_{j\in N} p_j}{m}\right\} \tag{1}$$

Zauważmy, że wartość C^* może być traktowana jako dolne oszacowanie optymalnej długości uszeregowania problemu $P \mid restricted\ k\text{-}pmtn \mid C_{\max}$ przy dowolnej wartości k.

3. Szeregowanie zadań częściowo podzielnych na dwóch procesorach

W tym rozdziale jest rozpatrywany szczególny przypadek problemu $P \mid restricted\ k\text{-}pmtn \mid C_{\max}$ gdy są dostępne tylko dwa procesory. Prezentowane tu właściwości i algorytmy będą wykorzystywane w rozdziale 4 do konstrukcji algorytmów dla ogólnego problemu $P \mid restricted\ k\text{-}pmtn \mid C_{\max}$ z dowolną liczbą procesorów.

Zauważmy, że problem szeregowania zadań na dwóch procesorach jest prosty, gdy $p_{\max} \geq \Sigma_{j\in N} p_j / m$. Zadanie z czasem obsługi $p_{\max}$ jest wykonywane wtedy na jednym procesorze, a pozostałe zadania na drugim. Inne łatwe przypadki zostały zidentyfikowane w pracy [5] i sformułowane w postaci poniższych twierdzeń.

Twierdzenie 1. *Jeżeli istnieje zadanie z czasem obsługi wynoszącym co najmniej 4k, to optymalne uszeregowanie problemu* $P2 \mid restricted\ k\text{-}pmtn \mid C_{\max}$ *można wyznaczyć w czasie O(n).*

Twierdzenie 2. *Jeżeli istnieją dwa zadania, jedno z czasem obsługi co najmniej 2k i drugie z czasem obsługi co najmniej 3k, to optymalne uszeregowanie problemu* $P2 \mid restricted\ k\text{-}pmtn \mid C_{\max}$ *można wyznaczyć w czasie O(n).*

Dowodząc powyższych twierdzeń, w pracy [5] przedstawiono też sposób wyznaczania optymalnych harmonogramów. W obu przypadkach uzyskuje się harmonogramy o długości C^*.

Wykorzystując powyższe właściwości, zaproponowano również następujący algorytm szeregowania zadań częściowo podzielnych na dwóch procesorach:

Algorytm 2DS

1. Jeżeli spełnione są założenia twierdzenia 1 lub 2 wyznacz optymalny harmonogram;
2. W przeciwnym przypadku:
   a) uszereguj zadania bez podziału stosując algorytm LPT,
   b) spróbuj zmniejszyć długość uszeregowania rozpatrując możliwości wymiany pary całych zadań (bez podziału) pomiędzy procesorami i wykonaj najlepszy wariant wymiany,
   c) spróbuj zmniejszyć długość uszeregowania poprzez wymianę najdłuższego zadania z jednego procesora (lub jego fragmentu) z każdym innym zadaniem bądź fragmentem z drugiego procesora;

Algorytm 2DS jest przerywany jeżeli w którymkolwiek z powyższych kroków uzyskuje się harmonogram z jednakowym obciążeniem obu procesorów. Wówczas mamy rozwiązanie optymalne. W pozostałych przypadkach nie ma gwarancji, że uzyskane rozwiązanie jest optymalne i należy traktować je jako przybliżone. Wykonując krok 2b) oraz 2c) szukamy w obu przypadkach tylko jednego – najlepszego wariantu wymiany i taki wariant realizujemy. Sposób podziału zadań na fragmenty, rozpatrywany w kroku 2c), został przedstawiony w pracy [5]. Poszczególne fragmenty podzielonych zadań mają długość nie mniejszą niż k. Żeby uniknąć konfliktów czasowych, pierwszy fragment podzielonego zadania umieszcza się na początku harmonogramu jednego procesora, a drugi fragment na końcu harmonogramu drugiego procesora. Dzięki takiemu rozmieszczeniu fragmentów tego samego zadania mamy gwarancję, że nie będą one wykonywane jednocześnie.

W eksperymentach obliczeniowych przedstawionych w rozdziale 5 badano też zmodyfikowany wariant algorytmu 2DS o nieco większym nakładzie obliczeń. Wariant ten będziemy nazywać 2DSM i różni się on od 2DS tylko krokiem 2b). W algorytmie 2DSM krok 2b) nie ogranicza się do wymiany tylko jednej pary zadań, ale operacja taka jest powtarzana wielokrotnie, jeżeli istnieje jeszcze możliwość dalszego zmniejszenia długości uszeregowania.

4. Szeregowanie zadań częściowo podzielnych na wielu procesorach

W przypadku gdy występują więcej niż dwa procesory proponujemy zastosowanie algorytmu dwuwarstwowego. Startujemy z uszeregowania wyznaczonego przez algorytm LPT bez podziału zadań. Następnie w kolejnych iteracjach warstwy nadrzędnej są wybierane dwa procesory z największym i najmniejszym obciążeniem. W warstwie podrzędnej, dla tych dwóch procesorów są stosowane algorytmy przedstawione w poprzednim rozdziale, tzn. 2DS lub 2DSM. W rezultacie, w poszczególnych iteracjach warstwy nadrzędnej dąży się do zmniejszenia długości uszeregowania najbardziej obciążonego procesora. Algorytm kończy się, gdy w pewnej iteracji nadrzędnej nie uzyskuje się już poprawy rozwiązania.

Istotną komplikacją w zastosowaniu takiego schematu szeregowania jest możliwość powstawania konfliktów czasowych przy wielokrotnym stosowaniu

algorytmu 2DS lub 2DSM dla tych samych procesorów. W celu uniknięcia tych konfliktów, czyli zapewnienia, że fragmenty tego samego podzielonego zadania nie będą jednocześnie wykonywane na kilku procesorach, proponowane są dwa podejścia.

W pierwszym podejściu, które będziemy nazywać FP+2DS, w każdej iteracji warstwy nadrzędnej wybiera się do wyrównywania obciążeń tylko te procesory, które wcześniej takiej operacji nie były poddawane. Schemat takiego algorytmu jest następujący:

Algorytm FP+2DS

1. $W := M$; {W – zbiór uwzględnianych procesorów (na początku wszystkie)}
2. Uszereguj zadania bez podziału stosując algorytm LPT;
3. Wybierz ze zbioru W najbardziej obciążony procesor P_{max} i najmniej obciążony P_{min};
4. Zastosuj algorytm 2DS dla procesorów P_{max} i P_{min};
5. Usuń procesory P_{max} i P_{min} ze zbioru W;
6. Jeżeli w kroku 4 zmniejszyła się długość uszeregowania C_{max} i $|W| \geq 2$, to idź do 3, w przeciwnym przypadku KONIEC;

W drugim podejściu, o nazwie TC+2DS, te same procesory mogą być wykorzystywane wielokrotnie, tzn. w warstwie nadrzędnej uwzględnia się wszystkie procesory (czyli zawsze $W = M$). Natomiast w kroku 4 bada się czy powtórne zastosowanie algorytmu 2DS do tego samego procesora nie spowoduje powstanie konfliktu czasowego. Jeżeli okaże się, że wystąpi konflikt, to taka wymiana zadań między procesorami nie jest dokonywana i dla algorytmu 2DS wybierany jest następny najmniej obciążony procesor P_{min}.

5. Eksperymentalne porównanie algorytmów

Efektywność algorytmów szeregowania zadań częściowo podzielnych była badana na serii przykładów testowych. Porównywano algorytmy TSRP3 i TSRP4 przedstawione w pracy [1] z proponowanymi algorytmami FP+2DS i TC+2DS. Ponadto badano też warianty FP+2DSM i TC+2DSM, w których jako procedurę podrzędną do wyrównywania obciążeń dwóch procesorów użyto 2DSM zamiast 2DS.

W tabeli 1 przedstawiono przykładowe wyniki eksperymentów obliczeniowych. Przedstawia ona średnie wartości otrzymanych długości uszeregowań C_{max} przy zastosowaniu poszczególnych algorytmów dla różnych wartości parametru k. Obliczenia zostały przeprowadzone dla 100 zestawów danych testowych i uzyskane wyniki uśredniono. W każdym zestawie danych było do uszeregowania 100 zadań na 11 równoległych identycznych procesorach. Czasy wykonywania poszczególnych zadań zostały wylosowane z przedziału [50, 150] zgodnie z rozkładem jednostajnym. W związku z tym dla $k > 75$ żadne z zadań nie jest podzielne.

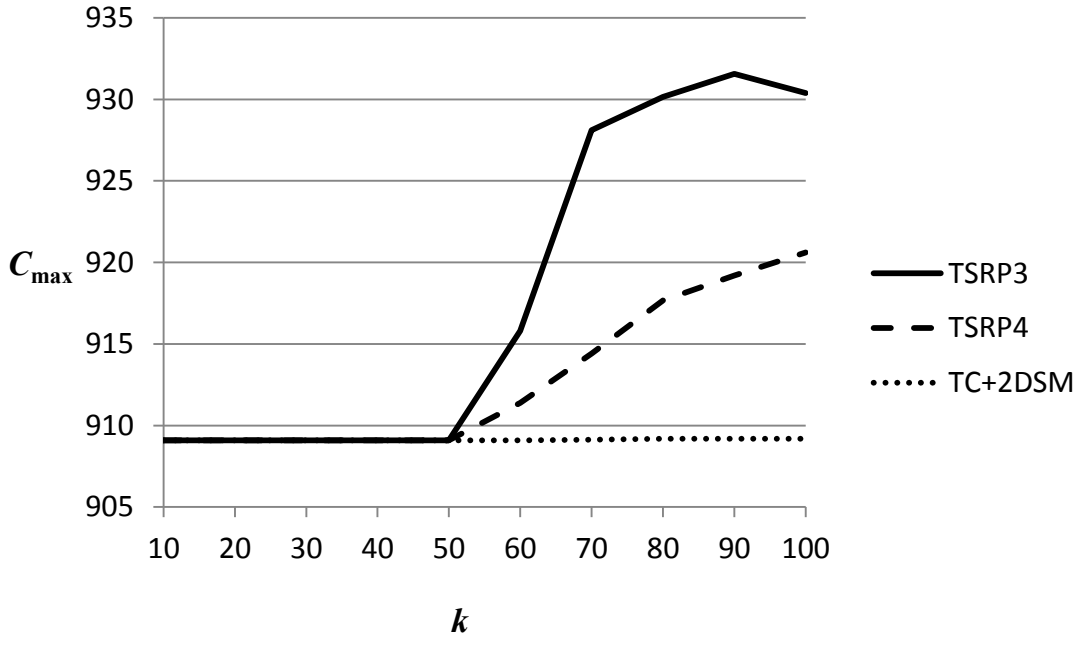
Porównując wyniki zamieszczone w tabeli 1 można zauważyć, że dla $k \leq 40$ wszystkie algorytmy poza FP+2DS szeregują zadania w sposób optymalny z jednakową długością uszeregowania na wszystkich procesorach, czyli $C_{max} = C^*$. Dla

$k > 40$ proponowane w pracy algorytmy zdecydowanie przewyższają algorytmy TSRP3 i TSRP4. Jest to wyraźnie widoczne na rys. 1, gdzie wyniki z tabeli 1 przedstawiono w formie wykresu. Różnice między uszeregowaniami uzyskanymi przy użyciu algorytmów FP+2DS, TC+2DS oraz FP+2DSM, TC+2DSM są nieznaczne i dlatego na rys. 1 przedstawiony wyniki tylko dla jednego z nich. Niemniej jednak można zauważyć, że procedura nadrzędna TC daje nieco lepsze wyniki niż FP, a ponadto lepsze wyniki daje zastosowanie algorytmu podrzędnego 2DSM zamiast 2DS. W rezultacie, spośród przedstawianych w pracy algorytmów najlepsze uszeregowania uzyskuje się stosując wariant TC+2DSM.

Tabela 1

Porównanie algorytmów szeregowania zadań częściowo podzielnych

	Długość uszeregowania (średnia ze 100 zestawów danych)					
k	TSRP3	TSRP4	FP+2DS	TC+2DS	FP+2DSM	TC+2DSM
10	909,08	909,08	909,09	909,08	909,08	909,08
20	909,08	909,08	909,09	909,08	909,08	909,08
30	909,08	909,08	909,09	909,08	909,08	909,08
40	909,08	909,08	909,09	909,08	909,08	909,08
50	909,09	909,09	909,09	909,08	909,08	909,08
60	915,81	911,39	909,09	909,08	909,08	909,08
70	928,11	914,40	909,63	909,24	909,18	909,12
80	930,14	917,67	909,63	909,63	909,18	909,18
90	931,56	919,19	909,63	909,63	909,18	909,18
100	930,38	920,62	909,63	909,63	909,18	909,18



Rys. 1. Porównanie algorytmów przy różnych wartościach parametru k

LITERATURA

1. Barański T.: Task scheduling with restricted preemptions. Proceedings of the Federated Conference on Computer Science and Information Systems, 2011, s. 231-238.
2. Błażewicz J., Ecker K.H., Pesch E., Schmidt G., Węglarz J.: Scheduling computer and manufacturing processes. Springer-Verlag, 1996.
3. Ecker K., Hirschberg R.: Task scheduling with restricted preemptions. Lecture Notes in Computer Science, vol. 694, Springer-Verlag, 1993, s. 464-475.
4. McNaughton R. : Scheduling with deadlines and loss functions. Management Science, vol. 6, 1959, s. 1-11.
5. Pieńkosz K., Prus A.: Task scheduling with restricted preemptions on two parallel processors. Proceedings of 20th International Conference on Methods and Models in Automation and Robotics (MMAR2015), 2015, s. 58-61.
6. Pinedo M.: Scheduling theory, algorithms, and systems. Prentice Hall, 2002.