ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2002
Seria: AUTOMATYKA z. 134 Nr kol. 1554

Jacek BŁAŻEWICZ[1, 2], Mikhail Y. KOVALYOV[3], Maciej MACHOWIAK[1],
Denis TRYSTRAM[4], Jan WĘGLARZ[1, 2]
[1] Politechnika Poznańska
[2] Instytut Chemii Bioorganicznej PAN w Poznaniu
[3] Institute of Engineering Cybernetics, National Academy of Sciences of Belarus, Mińsk
[4] Laboratory Informatique et Distribution, IMAG, Grenoble, Francja

SZEREGOWANIE ZADAŃ ELASTYCZNYCH

Streszczenie. Przedstawiono tutaj problem szeregowania *n* niezależnych zadań w systemie procesorów równoległych. Zadania są elastyczne, tj. zadanie może być wykonywane przez kilka procesorów jednocześnie oraz prędkość wykonywania zadania jest nieliniową funkcją od ilości procesorów przydzielonych do niego. Całkowita liczba procesorów w systemie wynosi *m* i jest to górna granica liczby procesorów, które mogą być używane przez wszystkie zadania w tym samym czasie. Dodatkowym założeniem jest podzielność zadań oraz możliwość zmiany liczby procesorów przydzielonych do zadania w trakcie jego wykonywania. Celem jest znalezienie uszeregowania, dla którego czas zakończenia wszystkich zadań jest najkrótszy z możliwych.
Prezentowany jest prosty algorytm o złożoności *O(n)*, rozwiązujący ten problem w przypadku, kiedy wszystkie funkcje prędkości wykonywania są wypukłe. Jeżeli funkcje te są wszystkie wklęsłe, przedstawiono algorytm pakowania prostokątów (PACK), który rozwiązuje ten problem w czasie wielomianowym.

SCHEDULING MALLEABLE TASKS

Summary. The problem of optimal scheduling n independent tasks on a parallel processor system is studied. The tasks are malleable, i.e. a task may be executed by several processors simultaneously and the processing speed of a task is a non-linear function of the number of processors allotted to it. The total number of processors is m and it is an upper bound on the number of processors that can be used by all the tasks simultaneously. It is assumed that the tasks are preemptable and the number of processors allotted to one task may change during its execution. The objective is to find a schedule for which the makespan, is minimized.
An *O(n)* algorithm is presented to solve this problem when all the processing speed functions are convex. If these functions are all concave the rectangles packing (PACK) algorithm is presented, which solves the problem in polynomial time.

1. Wprowadzenie

Szeregowanie *zadań elastycznych* (*malleable*) okazało się obiecującym narzędziem przy efektywnej implementacji pewnych praktycznych problemów pojawiających się w równoległych systemach komputerowych. Problem szeregowania takiego rodzaju zadań na procesorach równoległych możemy sformułować następująco [9, 10, 11, 12, 14].

Mamy n niezależnych zadań dostępnych w czasie początkowym $t = 0$. Należy je uszeregować do wykonania w systemie składającym się z m identycznych, równoległych procesorów. W każdej chwili t pewna liczba procesorów może zostać użyta do wykonania danego zadania. Jednakże procesor nie może wykonywać więcej niż jedno zadanie w danej jednostce czasu, a całkowita liczba używanych jednocześnie procesorów nie może być większa od m.

Z każdym zadaniem T_j związana jest pewna ilość $p_j > 0$ pracy do wykonania. Jeżeli do wykonania zadania T_j użyjemy, w przedziale czasowym o długości t, r procesorów, to ilość pracy do wykonania zadania wewnątrz tego przedziału jest równa $g_j(r)\ t$, gdzie $g_j(r) \geq 0$ jest niemalejącą funkcją prędkości wykonywania zadania, zdefiniowaną dla $r \in \{0, 1, ..., m\}$, oraz $g_j(0) = 0$ i $g_j(1) = 1$. Całkowita ilość pracy dla zadania T_j musi być równa p_j, dla każdego j.

Dla każdego zadania uszeregowanie specyfikuje przedziały czasowe, wewnątrz których dane zadanie powinno być wykonywane, oraz liczbę procesorów potrzebnych do wykonania zadania wewnątrz tych przedziałów. Celem jest znalezienie takiego uszeregowania, które spełnia powyższe ograniczenia oraz minimalizuje największy moment zakończenia wszystkich zadań (*makespan*).

Podobny problem, w którym procesory reprezentują ciągły, podzielny i odnawialny zasób, rozpatrywany był przez Węglarza [16]. Natomiast całą gamę problemów z zadaniami wieloprocesorowymi znajdziemy w [1, 2]. W dalszych rozważaniach będziemy rozróżniać oryginalny problem z dyskretnym (całkowitym) przydziałem zasobu (procesorów) jako P-DSCR oraz problem, w którym przydział procesorów nie musi być całkowity jako P-CNTN.

We wcześniejszych badaniach dużo uwagi poświecono problemowi P-FIX, w którym liczba procesorów przydzielonych do zadania jest niezmienna podczas jego wykonywania. Począwszy od Du i Leunga [7], którzy udowodnili NP-trudność tego problemu, wiele ciekawych rezultatów z tej dziedziny możemy znaleźć w [4, 5, 8, 9, 13, 14].

W naszych rozważaniach zajęliśmy się dwoma specjalnymi przypadkami: a) wszystkie funkcje prędkości wykonywania, od liczby przydzielonych procesorów są wypukłe (*convex*)

oraz b) wszystkie funkcje są wklęsłe (*concave*). Dla pierwszego przypadku prezentujemy algorytm rozwiązujący ten problem w czasie $O(n)$. Dla przypadku drugiego prezentujemy algorytm rozwiązujący problem P-CNTN (przydział procesorów do zadań nie musi być liczbą całkowitą) w czasie $O(n \max\{m, n \log^2 m\})$. Co więcej, zostało udowodnione, że minimalny moment zakończenia wszystkich zadań dla obu problemów, P-DSCR i P-CNTN, jest taki sam.

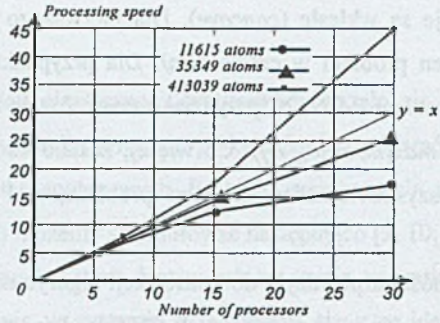
Ostatecznie zależność ta posłużyła do konstrukcji algorytmu pakowania prostokątów, który korzystając z rozwiązania problemu P-CNTN, konstruuje uszeregowanie dla oryginalnego problemu w czasie $O(n^2)$.

2. Motywacja

Poniżej przedstawiamy kilka rzeczywistych aplikacji z zadaniami równoległymi, w których znalazł zastosowanie rozpatrywany przez nas model.

2.1. Symulacja dynamiki molekuł

Symulacja dynamiki molekuł jest jedną z bardzo dynamicznie zmieniających się dziedzin nauki. Do sprawnego przeprowadzenia obliczeń ruchów atomów potrzeba zaawansowanych technik pozwalających na wyeliminowanie opóźnień komunikacyjnych, takich jak choćby asynchroniczne buforowanie opóźnień. Tylko w przypadku badania zachowania się białka potrzeba przeprowadzić obliczenia wzajemnych oddziaływań pomiędzy setkami a nawet tysiącami atomów [6]. Rzecz jasna, takie procesy potrzebują dużych zasobów pamięciowych. Na wielu najlepszych komputerach równoległych, takich jak Cray T3E, ze względu na uproszczenia sprzętowe i optymalizacje komunikacji, nie ma wirtualnego zarządzania pamięcią, z tego względu dostępna pamięć jest ściśle ograniczona. W związku z tym, kiedy instancja problemu nie mieści się w pamięci procesora, proces nie może być wykonany bezpośrednio. Do zakończenia procesu potrzeba „ręcznego" zarządzania pamięcią wirtualną, wykonując obliczenia poza jądrem (*out of core*), tj. obciążając dysk wykonaniem pośrednich obliczeń. Oczywiście, prowadzi to do zwiększenia czasu obliczeń. Dlatego też, kiedy liczba procesorów jest wystarczająca do przechowywania danych w pamięci podręcznej, możemy zaobserwować nadliniowy wzrost prędkości wykonywania. (patrz rys. 1).



Rys. 1. Funkcje prędkości wykonywania

Fig. 1. Processing speed functions

2.2. Rozkłady macierzowe

Jednym z takich przykładów zastosowań równoległych obliczeń bazujących na modelu zadań elastycznych jest rozkład dużych macierzy rzadkich Choleskiego (*LR*). Jeżeli założymy, że macierz nie mieści się w pamięci Cache, nastąpi spowolnienie obliczeń na jednym procesorze. W przeciwnym przypadku zaobserwowano, że zysk ze wzrostu prędkości wykonywania jest większy niż opóźnienia komunikacyjne spowodowane dodatkowymi procesorami. Prowadzi to do nadliniowej zależności między prędkością wykonywania a liczbą przydzielonych procesorów.

Z podobnymi problemami mamy do czynienia przy badaniu pływów oceanicznych, gdzie ruch mas wody zamodelowano za pomocą tysięcy połączonych ze sobą punktów tworzących przestrzenną kratę.

Model zadań elastycznych jest znakomitym i sprawnym narzędziem do rozwiązywania podobnych problemów, szczególnie w przypadku gdy, ze względu na rozmiar instancji, mamy kilka zadań i więcej procesorów.

3. Przydział zasobu ciągłego

Przytoczymy teraz podstawowe wyniki z dziedziny optymalnego przydziału zasobu ciągłego, które okazały się pomocne przy konstrukcji rozwiązania naszego problemu [16].

Podczas transformacji modelu ciągłego do naszych potrzeb wystąpił problem definiowania funkcji prędkości. wykonywania dla wartości niecałkowitych. Naturalnym podejściem było dążenie do zachowania takich własności funkcji, jak monotoniczność,

wklęsłość lub wypukłość. Własności te zostaną zachowane, kiedy rozszerzymy naszą funkcję g_j za pomocą następującej, odcinkami liniowej funkcji f_j:

$f_j(r) = \alpha_r g_j(\lfloor r \rfloor) + (1-\alpha_r) g_j(\lceil r \rceil)$ gdzie $\alpha_r = \lceil r \rceil - r$, $0 \le r \le m$, $j=1,\ldots,n$ lub

$f_j(r) = b_{j,\lceil r \rceil} r + d_{j,\lceil r \rceil}$ dla $0 \le r \le m$, $j=1,\ldots,n$.

Współczynniki $b_{j,\lceil r \rceil}$ i $d_{j,\lceil r \rceil}$ funkcji f_j można wyliczyć w czasie $O(mn)$. Zauważmy jednocześnie, że dla $r = 0,1,\ldots,m$, $f_j(r) = g_j(r)$. W dalszych rozważaniach problem P-DSCR z tak zdefiniowaną funkcją $f_j(r)$ będziemy nazywać problemem P-CNTN.

Zajmijmy się bliżej problemem P-CNTN, w którym niecałkowita liczba procesorów może być użyta do wykonania zadania oraz procesory reprezentują odnawialny, podzielny i ciągły zasób, którego ilość jest w danym czasie ograniczona przez m.

Niech $C^0_{\max}$ oznacza minimalną wartość $C_{\max}$ dla problemu P-CNTN. Korzystając z rozważań prowadzonych przez Węglarza [16] wprowadźmy następujące zbiory:

$$R = \{\mathbf{r} = (r_1,\ldots,r_n) \mid r_j \ge 0, \sum_{j=1}^{n} r_j \le m\}$$

zbiór *możliwych* (ze względu na problem P-CNTN) *przydziałów zasobu* oraz

$$U = \{\mathbf{u} = (u_1,\ldots,u_n) \mid u_j = f_j(r_j), j = 1,\ldots,n, r \in R\}$$

zbiór *możliwych, przekształconych przydziałów zasobu*. Dodatkowo wprowadźmy $\mathbf{p} = (p_1,\ldots,p_n)$.

Twierdzenie 1 (Węglarz [15, 16]). Niech $n \le m$, $convU$ oznacza powłokę wypukłą zbioru U oraz $\mathbf{u} = \mathbf{p} / C$ jest linią prostą w przestrzeni przekształconych przydziałów zasobu, daną w postaci parametrycznej $u_j = p_j / C$, $j = 1,\ldots,n$. Wtedy minimalną wartość $C^0_{\max}$ (makespan) dla problemu P-CNTN można znaleźć jako:

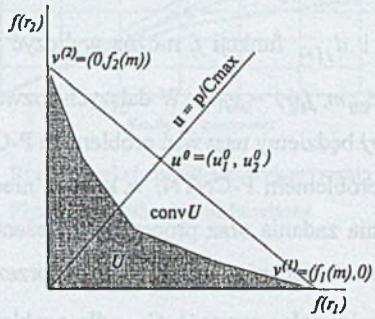
$$C^0_{\max} = \min\{C \mid C > 0, \mathbf{p} / C \in convU\} \tag{1}$$

Przedstawiając geometryczną interpretację Twierdzenia 1 możemy stwierdzić, że minimalna wartość $C^0_{\max}$ jest wyznaczona poprzez punkt przecięcia linii prostej $\mathbf{u} = \mathbf{p} / C$ z powłoką zbioru $convU$ w n-wymiarowej przestrzeni przekształconych przydziałów zasobu.

Oznaczmy taki punkt przecięcia przez $\mathbf{u}^0$, jednocześnie $C^0_{\max} = p_j / u^0_j$, $j = 1,\ldots,n$.

4. Wypukłe funkcje prędkości wykonywania

W przypadku kiedy wszystkie funkcje f_j prędkości wykonywania są dla każdego zadania wypukłe, zbiór U jest podzbiorem zbioru $convU$ (patrz rys.2).



Rys. 2. Zbiór U, $convU$ i linia u = p/C dla wypukłych funkcji prędkości wykonywania

Fig. 2. Set U, $convU$ and line u = p/C for convex processing speed functions

Rozważmy punkty $v^{(j)}$, takie że $v^{(j)}_j = f_j(m)$ i $v^{(j)}_l = 0, l \neq j, j = 1,\dots,n, l = 1,\dots,n$. Punkty $v^{(j)}$ reprezentują wszystkie wierzchołki płaszczyzny (powierzchni) zbioru $convU$ oraz punkt przecięcia $\mathbf{u}^0$ należy do tej płaszczyzny. Stąd $\mathbf{u}^0$ może być wyrażony jako liniowa kombinacja punktów $v^{(1)}, \dots, v^{(n)}$. Toteż dla $j = 1,\dots,n$ mamy:

$$u^0_j = \sum_{i=1}^{n} \lambda_i v^{(i)}_j = p_j / C^0_{\max} \text{ lub } p_j = \sum_{i=1}^{n} (\lambda_i C^0_{\max}) v^{(i)}_j, \text{ gdzie } \lambda_i \geq 0 \text{ i } \sum_{i=1}^{n} \lambda_i = 1.$$

Możemy teraz wyliczyć $\Delta_j = \lambda_j C^0_{max} = p_j / v^{(j)}_j = p_j / f_j(m)$. Ponieważ $\sum_{i=1}^{n} \Delta_i = C^0_{\max}$ oraz $p_j = \Delta_j f_j(m)$, optymalne uszeregowanie uzyskujemy poprzez wyznaczenie jednego przedziału czasowego długości Δ_j dla zadania T_j wraz z przydzieleniem do tego zadania wszystkich procesorów.

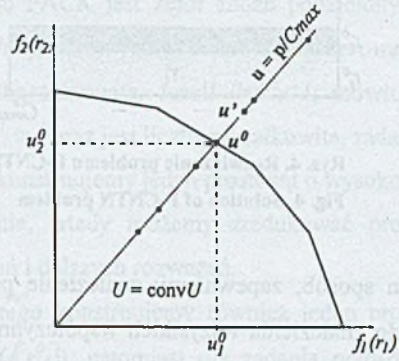
Optymalne uszeregowanie dla tego przypadku możemy znaleźć w czasie $O(n)$.

5. Wklęsłe funkcje prędkości wykonywania

Przedstawimy teraz rozwiązanie problemu P-CNTN dla wklęsłych funkcji prędkości wykonywania, które następnie wykorzystamy do konstrukcji rozwiązania problemu P-DSCR.

5.1. Rozwiązanie problemu P-CNTN

Dla tego przypadku zbiór U jest bryłą wypukłą w n-wymiarowej przestrzeni przekształconych przydziałów zasobu, co więcej zbiór $convU = U$ (patrz rys. 3).



Rys. 3. Punkt przecięcia u^0, linii $u = p/C$ oraz powłoki wypukłej zbioru U

Fig. 3. Intersection point u^0 of line $u = p/C$ and the boundary of convex polytope U

Do znalezienia rozwiązania problemu P-CNTN należy wykonać następujące czynności:

I. Rozważ zbiór I wszystkich punktów przecięcia płaszczyzn $u_j = f_j(r_j)$, $j = 1,...,n$; $r_j = 1,...,m$, z linią $\mathbf{u} = \mathbf{p} / C$, $C > 0$, w rozważanej n-wymiarowej przestrzeni.

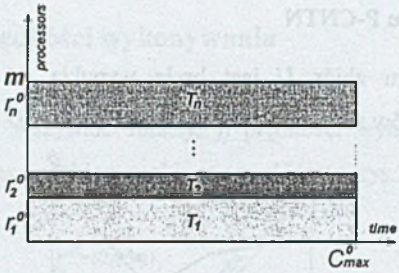
II. Stosując procedurę *bisekcji* znajdź taki punkt $\mathbf{u}' \in I$, $f^{-1}(\mathbf{u}') = \mathbf{r}'$, że $\sum_{j=1}^{n} r'_j \geq m$ oraz nie ma innych punktów pomiędzy $\mathbf{u}'$ i $\mathbf{u}^0$. Wynika stąd, że funkcja f_j nie posiada punktów nieciągłości w przedziale (r^0_j, r'_j), $j = 1,..,n$. Zatem, dla każdego $j = 1,...,n$, mamy:

$$u^0 = f_j(r^0_j) = b_{j,\lceil r'_j \rceil} r^0_j + d_{j,\lceil r'_j \rceil} = p_j / C^0_{max}$$

$$r^0_j = p_j / (b_{j,\lceil r'_j \rceil} C^0_{max}) - d_{j,\lceil r'_j \rceil} / b_{j,\lceil r'_j \rceil}$$

III. Z warunku $\sum_{j=1}^{n} r^0_j = m$ znajdź $C^0_{max} = (\sum_{j=1}^{n} p_j / b_{j,\lceil r'_j \rceil}) / (m + \sum_{j=1}^{n} d_{j,\lceil r'_j \rceil} / b_{j,\lceil r'_j \rceil})$, a następnie wylicz $\mathbf{u}^0$ oraz $\mathbf{r}^0$.

Ponieważ $p_j = u^0_j C^0_{max}$, $j = 1,...,n$, oraz $\sum_{j=1}^{n} r^0_j = m$, istnieje optymalne rozwiązanie problemu P-CNTN, w którym wszystkie zadania są wykonywane w przedziale czasowym $[0, C^0_{max}]$ i do zadania T_j przydzielono r^0_j procesorów, $j = 1,...,n$ (patrz rys. 4).



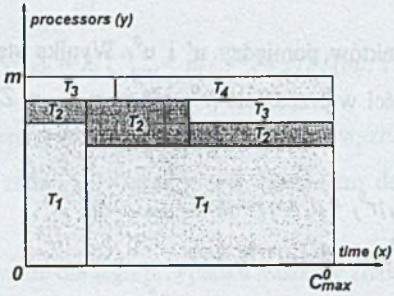
Rys. 4. Rozwiązanie problemu P-CNTN

Fig. 4. Solution of P-CNTN problem

Postępując w ten sposób, zapewniamy znalezienie przydziału zasobu r'_j w czasie $O(n^2 \log^2 m)$. Ponieważ do znalezienia wszystkich współczynników funkcji f_j potrzeba czasu rzędu $O(mn)$, dochodzimy do konkluzji, że problem P-CNTN można rozwiązać w czasie $O(n \max\{m, n \log^2 m\})$.

5.2. Rozwiązanie problemu P-DSCR

W optymalnym rozwiązaniu problemu P-CNTN każde zadanie jest wykonywane w przedziale czasowym $[0, C^0_{max}]$ i niecałkowita liczba r^0_i procesorów jest przydzielona do zadania T_i. Rozwiązanie problemu P-DSCR można przedstawić jako zbiór rozłącznych prostokątów w dwuwymiarowej przestrzeni (x, y) (patrz rys. 5).



Rys. 5. Rozwiązanie problemu P-DSCR

Fig. 5. Solution of P-DSCR problem

Tutaj oś (0, y) odpowiada liczbie procesorów przydzielonych do zadania, natomiast oś (0, x) jest osią czasu.

Każdy prostokąt jest powiązany z jakimś zadaniem, przy czym jedno zadanie może być reprezentowane przez kilka prostokątów.

Przedstawimy teraz zasadę działania *algorytmu (PACK) pakowania prostokątów* pozwalającego na optymalne rozwiązanie naszego problemu; w tym celu poczynimy kilka dodatkowych założeń i sformułujemy potrzebne lematy.

Instancją dla algorytmu PACK jest zbiór zadań podzielony na trzy podzbiory tzw. *zadań zerowych*, *pojedynczych* i *podwójnych*. Zadanie T_i jest *zerowe*, jeśli r^0_i (w rozwiązaniu problemu P-CNTN) jest liczbą całkowitą. Jeżeli $0<r^0_i<1$, mówimy, że T_i jest zadaniem *pojedynczym*, natomiast gdy $r^0_i>1$ oraz jest liczbą niecałkowitą, zadanie T_i jest *podwójne*.

Dla zadania *zerowego* konstruujemy jeden prostokąt o wysokości r^0_i i szerokości C^0_{max}. Jeśli pojawi się takie zadanie, wtedy możemy zredukować problem P-DSCR poprzez wykluczenie go ze zbioru zadań i dalszych rozważań.

Dla zadania *pojedynczego* konstruujemy również jeden prostokąt, ale o wysokości $\lceil r^0_i \rceil = 1$ i szerokości $\lambda_i p_i / f_i(\lceil r^0_i \rceil)$, natomiast dla zadania *podwójnego* dwa prostokąty o wysokościach odpowiednio $\lfloor r^0_i \rfloor$ i $\lceil r^0_i \rceil$ oraz szerokościach $\lambda_i p_i / f_i(\lfloor r^0_i \rfloor)$ i $(1-\lambda_i)p_i / f_i(\lceil r^0_i \rceil)$, gdzie $\lambda_i = (\lceil r^0_i \rceil - r^0_i) f_i(\lfloor r^0_i \rfloor) / f_i(r^0_i)$, $i = 1,\ldots, n$.

Następnie, bez przytaczania dowodów, wprowadzimy kilka pomocniczych lematów związanych z własnościami tak skonstruowanych prostokątów:

Lemat 1. Całkowite pole powierzchni wszystkich prostokątów jest równe $m\, C^0_{max}$.

Lemat 2. Szerokość prostokąta związanego z zadaniem *pojedynczym* nie przekracza C^0_{max}.

Lemat 3. Łączna szerokość obu prostokątów reprezentujących zadanie *podwójne* jest równa C^0_{max}.

Aby działać poprawnie algorytm PACK powinien spełniać następujące warunki:

a) Wszystkie prostokąty powinny się dokładnie mieścić w prostokącie, zwanym BIG (patrz rys. 6), o wysokości m i szerokości C^0_{max}. Ten warunek zachodzi, kiedy wszystkie zadania używają w każdym przedziale czasowym nie więcej niż m procesorów oraz czas zakończenia wszystkich zadań jest minimalny.

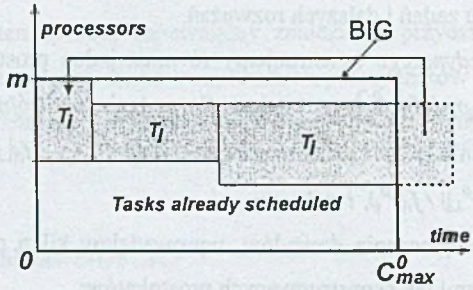
b) Wykonywanie na osi (0, x) prostokątów związanych z tym samym zadaniem może zostać przerwane tylko na końcu (prawy bok prostokąta BIG). Ten warunek zachodzi, gdy w każdym momencie wysokością prostokąta jest całkowita liczba procesorów przydzielonych w tym czasie do danego zadania.

c) Cała praca związana z zadaniem T_i zostanie wykonana.

Teraz zajmiemy się opisem kolejnych kroków algorytmu pakowania prostokątów. Algorytm ten składa się z *n* iteracji; w tym przypadku *n* oznacza liczbę wszystkich zadań

pojedynczych i *podwójnych*. Ponieważ zadania *zerowe* mają całkowity przydział procesorów, w ostatecznym rozwiązaniu będą wykonywane najpierw w każdym przedziale czasowym na stałej liczbie procesorów.

W każdej iteracji prostokąty odpowiadające zadaniom *pojedynczym* lub *podwójnym* są przydzielane (pakowane) w wolną przestrzeń prostokąta BIG, przy czym dla zadań *podwójnych* wyższy z prostokątów jest pakowany w pierwszej kolejności. Sposób, w jakim pakujemy kolejne prostokąty, nazwaliśmy *regułą południowo-zachodniego narożnika* i brzmi ona następująco: ***przydziel prostokąt najniższemu i położonemu najbardziej na lewo narożnikowi wolnej przestrzeni prostokąta BIG.***



Rys. 6. Przydział zadania podwójnego

Fig. 6. Assigning a 2-task

Algorytm PACK zatrzymuje się, kiedy wszystkie prostokąty zostały zapakowane. Można łatwo wykazać, że tak skonstruowany zbiór prostokątów spełnia warunki a) – c) oraz wyznacza optymalne rozwiązanie problemu P-DSCR. Zauważmy, że w rozwiązaniu optymalnym liczba przydzielonych do zadania T_i procesorów zmienia się co najwyżej dwa razy pomiędzy dwoma wartościami $\lfloor r^0_i \rfloor$ i $\lceil r^0_i \rceil$ (patrz rys. 5).

Do konstrukcji rozwiązania algorytm PACK potrzebuje czasu rzędu $O(n)$, co oznacza, że całkowitą złożoność czasową problemu P-DSCR szeregowania zadań elastycznych determinuje złożoność czasowa problemu P-CNTN, która wynosi $O(n \max\{m, n \log^2 m\})$.

6. Podsumowanie

Rozważaliśmy tutaj problem szeregowania zadań elastycznych z prędkościami wykonywania zależnymi nieliniowo od liczby przydzielonych procesorów. Pokazaliśmy, że ciągły przypadek problemu z przydziałem zasobu (procesorów), można rozwiązać w

wielomianowym czasie. Z drugiej strony, bardziej realistyczny problem dyskretny, rozwiązaliśmy w czasie $O(n)$ używając algorytmu pakowania prostokątów. Dalsze poszukiwania w tej dziedzinie koncentrują się na znalezieniu bardziej efektywnych rozwiązań dla problemu dyskretnego z dowolnymi funkcjami prędkości wykonywania.

LITERATURA

1. Błażewicz J., Drabowski M., Węglarz J.: Scheduling multiprocessor tasks to minimize schedule length, IEEE Trans. Comput., vol. C-35, 1986, s.389-393.
2. Błażewicz J., Ecker K., Plateau B., Trystram D. (eds.): Handbook on Parallel and Distributed Processing, Springer, Berlin, New York, 2000.
3. Błażewicz J., Ecker K., Pesch E., Schmidt G., Węglarz J.: Scheduling Computer and Manufacturing Processes, Springer, Berlin, New York, 1996.
4. Błażewicz J., Machowiak M., Mounie G., Trystram D., Węglarz J.: Scheduling malleable tasks with convex processing speed functions, Revista Iberoamericana de Computacion, vol. 2 (4), 2000, s.158-165.
5. Błażewicz J., Machowiak M., Mounie G., Trystram D.: Approximation algorithms for scheduling independent malleable tasks, In R. Sakellariou et al (Eds.): Euro-Par 2001, Lecture Notes in Computer Science, vol. 2150, Springer-Verlag 2001, s.191-197
6. Bernard P. E., Gautier T., Trystram D.: Large scale simulation of parallel molecular dynamics, In Proceedings of Second Merged Symposium IPPS/SPDP 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, San Juan, Puerto Rico, 1999.
7. Du J., Leung J.Y-T.: Complexity of scheduling parallel tasks systems, SIAM Journal on Discrete Mathematics, vol. 2, 1989, s.473-487.
8. Drozdowski M.: Scheduling multiprocessor tasks - An overview, European Journal of Operational Research, vol. 94, 1996, s.215-230.
9. Ludwig W. T.: Algorithms for scheduling malleable and nonmalleable parallel tasks, PhD thesis, University of Wisconsin - Madison, Department of Computer Science, 1995.
10. Rapine C., Scherson I., Trystram D.: On-line scheduling of parallelizable jobs, Lecture Notes in Computer Science, vol. 1470, 1998.
11. Scherson I., Subramanian R., Reis V., Campos L.: Scheduling computationally intensive data parallel programs, In Ecole francaise de parallelisme, reseaux et systemes, Placement dynamique et repartition de charge: application aux systemes paralleles et repartis, Presquele de Giens, INRIA, 1996, s.107-129.
12. Ludwig W., Schwiegelshohn U., Turek J., Wolf J., Yu P.: Smart SMART bounds for weighted response time scheduling, SIAM Journal on Computing, vol. 28, 1999, s.237-253.

13. Mounie G., Rapine C., Trystram D.: Efficient approximation algorithms for scheduling malleable tasks, In Eleventh ACM Symposium on Parallel Algorithms and Architectures (SPAA'99), ACM, 1999, s.23-32.

14. Turek J., Wolf J., Yu P.: Approximate algorithms for scheduling parallelizable tasks, In 4th Annual ACM Symposium on Parallel Algorithms and Architectures, 1992, s.323-332.

15. Węglarz J.: Project scheduling with continuously-divisible, doubly constrained resources, Management Sci., vol. 27, 1981, s.1040-1052.

16. Węglarz J.: Modelling and control of dynamic resource allocation project scheduling systems, In S. G. Tzafestas (ed.), Optimization and Control of Dynamic Operational Research Models, North-Holland, Amsterdam 1982.

Recenzent: Prof. Adam Janiak

Abstract

Scheduling *malleable tasks* has been proved to be a promising way for efficient implementation of some practical problems on parallel and distributed computers. The problem of scheduling malleable tasks on parallel processors can be formulated as follows. [9, 10, 11, 12, 14]

There are n independent tasks that are available at time zero. They are to be scheduled for processing in a system consisting of m, identical *parallel processors*. We will use natural numbers to denote tasks and processors. At each time instant, any number of processors can be used to execute a task. However, no processor can handle more than one task at a time and the total number of processors executing the tasks should not exceed m at any time.

An amount $p_j > 0$ of work is associated with each task T_j. If r processors are used to execute task T_j in a time interval of length t, then the amount of work for computing this task within this interval is equal to $g_j(r)\ t$, where $g_j(r) \geq 0$ is a non-decreasing *processing speed function* defined for $r \in \{0, 1, \ldots, m\}$, $g_j(0) = 0$, $g_j(1)=1$. The total amount of work done on task T_j must be equal to p_j, $j = 1, \ldots, n$.

For each task, a schedule specifies the time intervals within which this task is executed and the numbers of processors allocated to the task within these intervals. The objective is to find a schedule which satisfies the above constraints and such that the maximum task completion time, i.e. the *makespan* C_{max}, is minimized.

Denote the minimum C_{max} value by C^*_{max}.

A motivation for this problem comes from the optimal scheduling of multiprocessor systems enabling for an execution of large scale parallel computations. A similar problem in which the processors represent a continuously divisible renewable resource bounded from above, has been studied by Węglarz [16] (cf. also 3).

In the sequel, we refer to our original problem with a discrete (integer) resource as problem P-DSCR and to the problem where the processor (resource) allocation is not required to be integer (i.e. with a continuously divisible resource) as problem P-CNTN. In problem P-CNTN, functions $g_j(r)$ are interpolated by piecewise linear functions between integer points.

Observe that in problem P-DSCR the number of processors allocated to a task may change during the task execution. A malleable task my be interrupted and the processors allocated to it can be redistributed during the execution. It means that the parts of one task can be executed in adjoining time intervals on the different number of processors. The cost of preemption will be included into the speed functions.

The following results are available for problem P-FIX in which the (integer) number of processors allocated to a task cannot change [4, 5, 8, 9, 13, 14]. Du and Leung [7] proved that this problem is NP-hard. Drozdowski [8] provided a survey of the complexity and algorithms for multiprocessor task scheduling problems (see also [2]).

In this paper, we distinguish two special cases of the problem: a) the dependence of all processing speed functions on a number of processors allotted is convex and b) this dependence is concave. For case a), an $O(n)$ algorithm is presented to solve both problems P-CNTN and P-DSCR. For case b), problem P-CNTN is shown to be solvable in $O(n \max\{m, n \log^2 m\})$ time. Moreover, it is proved that the minimum makespan values for problems P-DSCR and P-CNTN coincide. The rectangles packing algorithm is presented, which using a solution for P-CNTN problem constructs the solution for original problem in $O(n)$ time.