RÓWNOLEGŁE WYZNACZANIE WARTOŚCI FUNKCJI CELU PROBLEMU PRZEPŁYWOWEGO

Wojciech BOŻEJKO, Mariusz UCHROŃSKI

Streszczenie: Obliczenia wartości funkcji celu złożonych problemów optymalizacji dyskretnej stanowią zwykle najbardziej czasochłonny moduł algorytmu optymalizacyjnego. Pozostałe czynności - związane z zarządzaniem - zajmują statystycznie 1-3% czasu trwania iteracji algorytmu poszukiwań lokalnych, takiego jak poszukiwanie z zabronieniami (t*abu search*), symulowane wyżarzanie czy poszukiwanie rozproszone (s*catter search*), zatem ich przyspieszanie poprzez realizację w środowisku równoległym daje znikomo mało korzyści. Odmiennie, przyspieszenie działania obliczeń numerycznych, takich jak obliczanie wartości funkcji celu lub przeglądanie otoczenia, poprzez zaimplementowanie ich w środowisku wieloprocesorowym może znacznie poprawić wydajność działania algorytmu przeszukującego przestrzeń rozwiązań.

Słowa kluczowe: algorytm lokalnych poszukiwań, permutacyjny problem przepływowy, przyspieszenie, algorytm równoległy, pamięć współdzielona.

1. Wprowadzenie

Metaheurystyki oparte na metodzie lokalnych poszukiwań mogą być przedstawione jako procesy badania grafu, w którym wierzchołki stanowią punkty przestrzeni rozwiązań (np. permutacje), a łuki odpowiadają relacji sąsiedztwa łączą wierzchołki będące rozwiązaniami sąsiednimi w tej przestrzeni. Poruszanie się po takim grafie definiuje pewną *drogę* (lub inaczej *trajektorię*). Równoległe algorytmy metaheurystyczne używają wielu procesorów do współbieżnego generowania lub przeglądania grafu sąsiedztwa.

Można zdefiniować dwa podejścia do zrównoleglania procesu lokalnego poszukiwania, w zależności od ilości trajektorii badanych współbieżnie w grafie sąsiedztwa.

- pojedyncza trajektoria: algorytmy drobnoziarniste (tj. z częstą komunikacją),
- wiele trajektorii: algorytmy gruboziarniste (gdzie komunikacja nie jest zbyt częsta).

Podejścia te stawiają przed algorytmem pewne wymagania dotyczące częstotliwości komunikacji, co implikuje rodzaj ziarnistości. Algorytmy drobnoziarniste odpowiadają podejściu z częstszą komunikacją, gruboziarniste - z rzadszą.

Algorytmy jednościeżkowe, wyznaczające pojedynczą trajektorię przejścia przez przestrzeń rozwiązań, mogą czynić to współbieżnie poprzez podział procesu badania otoczenia na kilka procesorów, z których każdy bada pewną część otoczenia szukając najlepszego ruchu. Idea ta została zaproponowana najwcześniej dla sekwencyjnych algorytmów poszukiwań, patrz Nowicki i Smutnicki [9] pod nazwą metody reprezentantów (*representatives*). Pochodzenie nazwy jest ściśle związane z działaniem metody, bowiem z każdej części otoczenia zostaje wybrany reprezentant, a dopiero spośród reprezentantów najlepszy jako następny punkt trajektorii poszukiwań. Zrównoleglona może być także proces obliczania wartości funkcji celu aktualnie badanego punktu przestrzeni rozwiązań. Często jest to na przykład algorytm wyznaczania najdłuższej drogi w pewnym grafie, lub też maksymalnego czy minimalnego przepływu.

Równoległe obliczenia dla jednej trajektorii. Celem tej metody jest po prostu przyspieszenie przejścia grafu sąsiedztwa poprzez zrównoleglenie najbardziej czasochłonnych operacji - czyli obliczania wartości funkcji celu, bądź zrównoleglenie procesu generowania sąsiadów. W przypadku zrównoleglania obliczania wartości funkcji celu przyspieszenie obliczeń może być uzyskane przy zachowaniu identycznej trajektorii przejścia przez graf, jak trajektoria algorytmu sekwencyjnego. W drugim przypadku - dekompozycji generowania otoczenia na procesory równoległe - zaistnieć może sytuacja, w której algorytm, sprawdzając równolegle większą ilość sąsiadów niż to czyni wersja sekwencyjna (najczęściej zaopatrzona w mechanizm redukcji rozmiarów otoczenia), poruszać się będzie po trajektorii lepszej niż sekwencyjny odpowiednik, wyznaczając korzystniejszą trasę przejścia przez graf i tym samym dochodząc do lepszych rezultatów obliczeń (wartości funkcji celu).

Zrównoleglanie przeszukiwania jednościeżkowego posiada średnią lub drobną ziarnistość i wymaga częstej komunikacji i synchronizacji. Jest ono ściśle związane z charakterystyką rozwiązywanego problemu i zdefiniowanej struktury sąsiedztwa. Pierwsze aplikacje bazujące na opisywanym modelu pojawiły się w kontekście zrównoleglenia metody symulowanego wyżarzania i algorytmu genetycznego. Chociaż równoległa dekompozycja sąsiedztwa nie zawsze prowadzi do redukcji czasu obliczeń, jest często stosowana do zwiększania rozpatrywanego sąsiedztwa. Tego typu algorytm równoległy *tabu search* dla problemu komiwojażera został zaproponowany przez Fiechtera [8]. Synchroniczny tabu serach był także badany przez Porto i Ribeiro [10, 11]. Bożejko [3] zaproponował równoległą jednościeżkową wersję algorytmu *dynasearch* dla problemu jednomaszynowego.

Aarts i Verhoeven [1] różnicują klasę jednościeżkowych algorytmów równoległego przeszukiwania na dwie podklasy. Klasa jednokrokowa (*single - step*) obejmuje algorytmy, w których badanie otoczenia jest dzielone pomiędzy równoległe procesory, ale jako wynik wybierany jest jeden ruch. W klasie wielokrokowej (*multiple - step*) sekwencja kolejnych ruchów w grafie sąsiedztwa jest wykonywana współbieżnie.

Przegląd zagadnień związanych z równoległymi metaheurystycznymi algorytmami jedno- i wielościeżkowymi znaleźć można w pracach Bożejko, Pempera, Smutnicki [4,5] oraz Bożejko, Wodecki [6].

2. Permutacyjny problem przepływowy

Opis klasycznego permutacyjnego problemu przepływowego znaleźć można w pracach [4,5,6,9]. Niech $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ będzie permutacją zadań $\{1,2,\ldots,n\}$, a Π zbiorem wszystkich takich permutacji. Każda permutacja $\pi \in \Pi$ wyznacza jednoznacznie kolejność wykonywania zadań na maszynach (na każdej maszynie taką samą). Do wyznaczenia terminów zakończenia wykonywania zadań C_{ij} używa się wzoru rekurencyjnego

$$C_{ij} = \max\{C_{i-1,\pi(j)}, C_{i,\pi(j-1)}\} + p_{i\pi(j)}, \qquad (1)$$
$$i = 1,2,\ldots,m,\ j = 1,2,\ldots,n,$$

z warunkiem początkowym

$$C_{i\pi(0)} = 0,\ i = 1,2,\ldots,m,\ C_{0\pi(j)} = 0,\ j = 1,2,\ldots,n,$$

lub nierekurencyjnego

$$C_{i\pi(j)} = \max_{1=j_0 \le j_1 \le \ldots \le j_i = j} \sum_{s=1}^{i} \sum_{j=j_{i-1}}^{j_i} p_{s\pi(j)} \,. \qquad (2)$$

Ponieważ złożoność obliczeniowa metody ze wzoru (1) jest $O(mn)$, zaś tej ze wzoru (2) jest $O\left(\left(\frac{j+i-2}{i-1}\right)(j+1-1) \right) = O\left(\frac{(n+m)^{n-1}}{(n-1)!} \right)$, w praktyce używany jest wyłącznie pierwszy z nich. Drugi wzór jest używany tylko do dowodzenia własności.

W teorii szeregowania rozpatrywane są najczęściej dwa NP-trudne problemy $F^* \parallel f_{\max}$ oraz $F^* \parallel \sum f_i$. W obu należy wyznaczyć permutację $\pi^* \in \Pi$ taką, że:

$$F(\pi^*) = \min_{\pi \in \Pi} F(\pi)$$

gdzie

$$F(\pi) = \sum_{j=1}^{n} f_{\pi(j)}(C_{m\pi(j)}), \;\; F(\pi) = \max_{1 \le j \le n} f_{\pi(j)}(C_{m\pi(j)})$$

dla $\gamma \in \{\sum f_i, f_{\max}\}$ odpowiednio. Dla $r_j = const$ problem ten jest równoważny temu z kryterium średniego czasu przepływu.

Model grafowy. Wielkość C_{ij} ze wzorów (1) i (2) można także wyznaczyć posługując się modelem grafowym problemu. Dla danej kolejności wykonywania zadań $\pi \in \Pi$ tworzymy graf $G(\pi) = (M \times N, F^0 \cup F^*)$, gdzie $M = \{1,2,\ldots m\}$, $N = \{1,2,\ldots n\}$.

$$F^0 = \bigcup_{s=1}^{m-1} \bigcup_{t=1}^{n} \{((s,t),(s+1,t))\}$$

jest zbiorem łuków technologicznych (pionowych), zaś

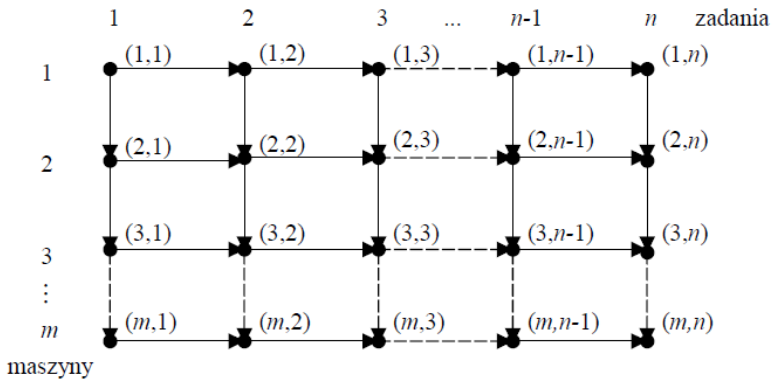
$$F^0 = \bigcup_{s=1}^{m} \bigcup_{t=1}^{n-1} \{((s,t),(s,t+1))\}$$

jest zbiorem łuków kolejnościowych (poziomych). Łuki grafu $G(\pi)$ nie posiadają obciążeń, natomiast obciążenie każdego węzła (s,t) wynosi $p_{s,\pi(t)}$. Czas C_{ij} zakończenia wykonywania zadania $\pi(j)$, $j = 1,2,\ldots,n$, na maszynie i, $i = 1,2,\ldots,m$, odpowiada długości najdłuższej ścieżki prowadzącej z wierzchołka $(1,1)$ do wierzchołka (i,j), wraz z obciążeniem tego ostatniego. Dla problemu $F^* \parallel C_{\max}$, wartość funkcji kryterialnej dla ustalonego π jest równa długości ścieżki krytycznej w grafie $G(\pi)$.

3. Poszukiwanie jednowątkowe

Metody te dopuszczają istnienie tylko jednego procesu (wątku) zarządzającego poszukiwaniami. Proces ten wykonuje cyklicznie iteracje, na które składają się:

- obliczenia numeryczne (np. wyznaczanie wartości funkcji celu),
- funkcje zarządzania (np. wybór rozwiązania, realizacja pamięci obliczeń, akceptacja rozwiązań).



Rys. 1. Graf $G(\pi)$

Czynności związane z zarządzaniem zajmują statystycznie 1-3% czasu trwania iteracji, zatem ich przyspieszanie poprzez realizację w środowisku równoległym daje znikomo mało korzyści. Odmiennie, przyspieszenie działania obliczeń numerycznych poprzez zaimplementowanie ich w środowisku wieloprocesorowym może znacznie poprawić wydajność działania algorytmu przeszukującego przestrzeń rozwiązań. Ważny jest przy tym aspekt teoretyczny przyspieszenia - użyte metody i własności powinny prowadzić do powstania algorytmów kosztowo optymalnych, czyli takich, których koszt rozwiązania problemu w systemie równoległym (iloczyn czasu wykonania i ilości użytych procesorów) jest proporcjonalny do czasu wykonania najszybszego znanego algorytmu rozwiązującego ten problem na jednym procesorze. Dla pewnych problemów uzyskanie takiej własności może być trudne - zaakceptować wówczas można także metody nie będące kosztowo optymalnymi, jeśli dają one duży zysk czasowy. Innym zagadnieniem może też być stworzenie metod, nastawionych wyłącznie na uzyskanie maksymalnego zysku związanego ze skróceniem czasu obliczeń - bez względu na liczbę użytych procesorów (czyli stawiające tezę: ,,mając dowolną ilość procesorów można obliczyć ... w czasie ..."). Warto zwrócić uwagę, że co prawda maksymalna liczba równolegle pracujących procesorów w największych współczesnych systemach obliczeń równoległych jest rzędu 10^6 procesorów, lecz metody obliczeń biologicznych dopuszczają równoległe wykorzystanie nawet 10^{23} procesorów. Procesory takie są wyspecjalizowanymi cząstkami (np. DNA lub białek), pozwalającymi wykonywać specjalnie spreparowane algorytmy służące do rozwiązywania problemów optymalizacji kombinatorycznej. Metoda ta znalazła już swą praktyczną realizację (rozwiązanie problemu komiwojażera za pomocą rekombinacji DNA, Adleman [2]). Tak więc w kontekście obliczeń biologicznych stworzenie szybkich metod, bazujących nawet na ogromnych ilościach procesorów, może być praktycznie użyteczne.

W analizie działania algorytmów poszukiwania można wyróżnić następujące elementy związane z oceną rozwiązań:

- obliczanie wartości funkcji oceny pojedynczego rozwiązania (np. algorytm symulowanego wyżarzania),
- obliczanie wartości funkcji oceny dla grupy rozwiązań sąsiednich (np. algorytm poszukiwania za zabronieniami),

- obliczanie wartości funkcji oceny dla grupy rozwiązań rozproszonych (np. algorytmy genetyczne, poszukiwania rozproszonego).

Omówimy i zanalizujemy bardziej szczegółowo punkt pierwszy to jest równoległe obliczanie wartości funkcji celu danego rozwiązania.

W dowodach twierdzeń dotyczących złożoności obliczeniowej algorytmów równoległych wykorzystamy następujące powszechnie znane fakty dotyczące obliczeń na teoretycznym modelu komputera równoległego PRAM:

Fakt 1 *Ciąg sum prefiksowych* $(y_1, y_2, \ldots, y_n)$ *ciągu wejściowego* $(x_1, x_2, \ldots, x_n)$ *takich, że*

$$y_k = y_{k-1} + x_k = x_1 + x_2 + \ldots + x_k \text{ dla } k = 2,3,\ldots,n$$

gdzie $y_1 = x_1$ *można wyznaczyć na maszynie EREW PRAM w czasie* $O(\log n)$ *za pomocą* $O(n/\log n)$ *procesorów.*

Fakt 2 *Sumę ciągu wejściowego* $(x_1, x_2, \ldots, x_n)$ *można wyznaczyć na maszynie EREW PRAM pojedynczego w czasie* $O(\log n)$ *za pomocą* $O(n/\log n)$ *procesorów.*

Fakt 3 *Wartość minimalna pojedynczego maksymalna ciągu wejściowego* $(x_1, x_2, \ldots, x_n)$ *można wyznaczyć na maszynie EREW PRAM pojedynczego czasie* $O(\log n)$ *za pomocą n procesorów.*

Fakt 4 *Wartość minimalną pojedynczego maksymalną ciągu wejściowego* $(x_1, x_2, \ldots, x_n)$ *można wyznaczyć na maszynie EREW PRAM za pomocą* $O(n/\log n)$ *pojedynczego czasie* $O(\log n)$.

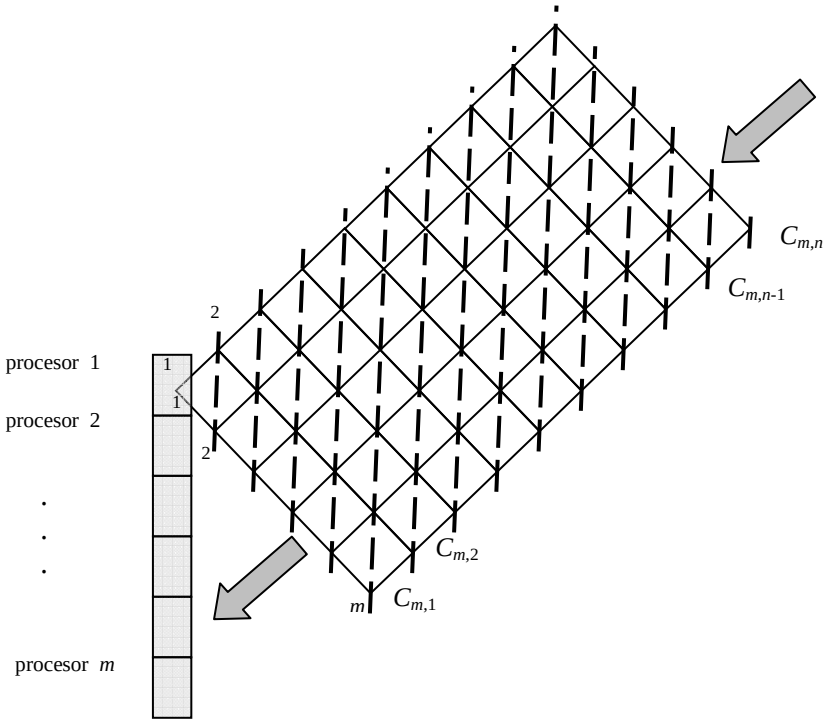
Fakt 5 *Wartość* $y = (y_1, y_2, \ldots, y_n)$ *gdzie* $y_i = f(x_i)$, $x = (x_1, x_2, \ldots, x_n)$ *można policzyć na maszynie CREW PRAM pojedynczego czasie* $O(c) = O(1)$ *na n procesorach, gdzie c jest czasem potrzebnym do wyznaczenia pojedynczej wartości* $y_i = f(x_i)$.

Fakt 6 *Zagadnienie sformułowane w Fakcie 5 można rozwiązać w czasie* $O(\log n)$ *na* $O(n/\log n)$ *procesorach.*

Fakt 7 *Jeśli algorytm* A *działa na p – procesorowej maszynie PRAM w czasie pojedynczego, to dla każdego* $p' < p$ *istnieje algorytm* A' *dla tego samego problemu działający na* p' *procesorowej maszynie PRAM w czasie* $O(pt/p')$.

Fakty 1 - 4 i 7 można znaleźć na przykład w pracy Cormen i in. [7]. Twierdzenia zawarte w dalszej części tego rozdziału zostały sformułowane dla modelu CREW (*Concurrent Read Exclusive Write*) maszyny PRAM, to jest z możliwością równoległego odczytu komórek pamięci (równoległego zabronieniem równoległego zapisu). Ułatwiło to przedstawienie idei algorytmów zawartych w dowodach, choć część twierdzeń prawdziwa jest także dla modelu EREW (*Exclusive Read Exclusive Write*) z sekwencyjnym dostępem do komórek pamięci.

Twierdzenie 1 *Dla ustalonej permutacji* π, *wartość kryterium w problemach* $F^* \parallel \sum f_i$, $F^* \parallel f_{\max}$ *można wyznaczyć na maszynie CREW PARAM w czasie* $O(n+m)$ *wykorzystując m procesorów.*

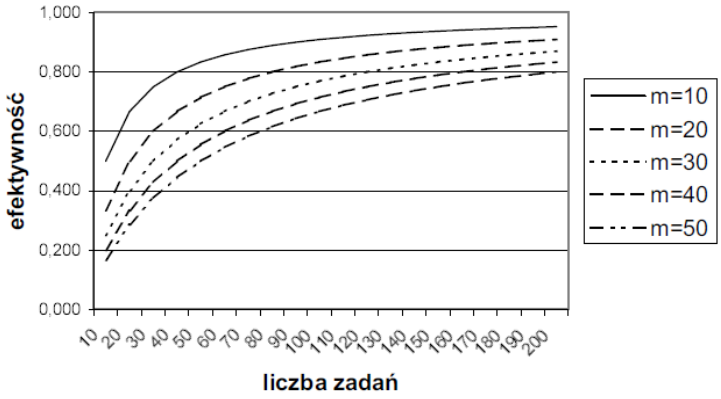


Rys. 2. Kolejność obliczania C_{ij} przez *m* procesorów**.**

Dowód. Bez straty ogólności możemy przyjąć, że $\pi = (1,2,\ldots n)$. Skorzystamy z zależności rekurencyjnej (1). Proces równoległego wyznaczania wartości C_{ij} wykonujemy następująco. W pierwszym takcie procesor 1 wyznacza wartość $C_{1,1}$. W drugim takcie procesory 1 i 2 wyznaczają równolegle wartości odpowiednio $C_{2,1}$ i $C_{1,2}$. W takcie k – tym procesor numer t wyznacza wartość $C_{k-t+1,t}$ jeśli $1 \le k-t+1 \le n$. Opisywana kolejność jest przedstawiona na Rys. 2 za pomocą linii przerywanych, na tle grafu $G(\pi)$, zdefiniowanego w Rozdz. (w dalszej części pracy ten charakterystyczny graf będziemy nazywać grafem siatkowym).

Opisane postępowanie wymaga wykonania $n+m-1$ kroków. Dla obliczenia wartości kryterium $\sum f_i = \sum_{j=1}^{n} f_j(C_{m,j})$ należy jeszcze dodać *n* wartości $f_j(C_{m,j})$, co można wykonać sekwencyjnie w *n* iteracjach lub równolegle, za pomocą *m* procesorów przy złożoności $O(n/m + \log n)$. Ostatecznie złożoność obliczeniowa wyznaczenia kryterium w problemie $F^* \parallel \sum f_i$ wynosi więc $O(n+m)$ przy pomocy m procesorów.

Wniosek 1 Przyspieszenie metody bazującej na Tw. 1 wynosi $O\left(\frac{nm}{n+m}\right)$, efektywność wynosi $O\left(\frac{n}{n+m}\right)$.



Rys. 3. Efektywność metody zaproponowanej w Tw. 1.

Pokażemy dalej inna metodę, dla problemów $F^* \| \sum f_i$, $F^* \| f_{\max}$, o zdecydowanie lepszej charakterystyce.

Twierdzenie 2 *Dla ustalonej permutacji* π, *wartość kryterium w problemach* $F^* \| \sum f_i$, $F^* \| f_{\max}$ *można wyznaczyć w czasie* $O(n+m)$ *na* $O\left(\frac{nm}{n+m}\right)$ *procesorowej maszynie CREW PRAM.*

Dowód. Bez straty ogólności możemy przyjąć, że $\pi = (1,2,\ldots n)$. Opieramy się na schemacie obliczeń zaprezentowanym na Rys. . Niech $p \le m$ będzie ilością użytych procesorów. Proces obliczeń będzie przeprowadzany dla *poziomów* $k = 1,2,\ldots, n+m-1$. Na poziomie k przeprowadzane są obliczenia wszystkich wartości $C_{i,j}$ takich, że $i+j-1=k$. Wszystkie obliczenia na poziomie k przeprowadzane są po obliczeniu wszystkich wartości $C_{i,j}$ na poziomie $k-1=k$. Głębokość całego schematu obliczeń (ilość poziomów) niech oznaczona będzie przez d i wynosi $d = n+m-1$

Niech n_k będzie liczbą wartości $C_{i,j}$ obliczanych na poziomie *k*. Wobec tego

$$\sum_{k=1}^{d} n_k = nm,$$

ponieważ ilość wszystkich obliczanych wartości $C_{i,j}$ na wszystkich poziomach wynosi *nm*. Rozważmy n_k elementów na poziomie *k*. Grupujemy je w $\lceil n_k / p \rceil$ grup, z których pierwsze $\lfloor n_k / p \rfloor$ zawiera po *p* elementów, a pozostałe elementy (jest ich najwyżej *p*) niech należą do ostatniej grupy. Obliczenia równoległe maszyny PRAM na *k*- tym poziomie wykonywane są więc w czasie $O(\lceil n_k / p \rceil)$. Łączny czas obliczeń jest równy sumie czasów wykonywania na wszystkich poziomach i jest rzędu

$$\sum_{k=1}^{d} \left\lceil \frac{n_k}{p} \right\rceil \le \sum_{k=1}^{d} \left(\frac{n_k}{p} + 1 \right) = \frac{nm}{p} + d = \frac{nm}{p} + n + m - 1 \cdot$$

Przyjmując $p = O\left(\frac{nm}{n+m} \right)$, łączny czas obliczeń wartości $C_{i,j}$ wyniesie $O(n+m)$.

Obliczenie wartości kryterium $\sum f_i = \sum_{j=1}^{n} f_j(C_{m,j})$ można wykonać za pomocą jednego procesora w czasie $O(n)$. Zauważmy, że można wykonać to szybciej, za pomocą większej ilości procesorów, ale w rozpatrywanym przypadku, chcąc uzyskać złożoność $O(n+m)$, wystarczy to zrobić sekwencyjnie. Ostatecznie łączny czas obliczeń wyniesie $O(n+m)$.

Wniosek 2 Przyspieszenie metody bazującej na Tw. 2 wynosi $O\left(\frac{nm}{n+m} \right)$. Koszt wynosi $O(nm)$. Ilość użytych procesorów $O\left(\frac{nm}{n+m} \right) = O\left((n^{-1} + m^{-1})^{-1} \right)$ jest rzędu średniej harmonicznej liczb *n* i *m*. Zaproponowana metoda jest kosztowo optymalna.

4. Eksperymenty obliczeniowe

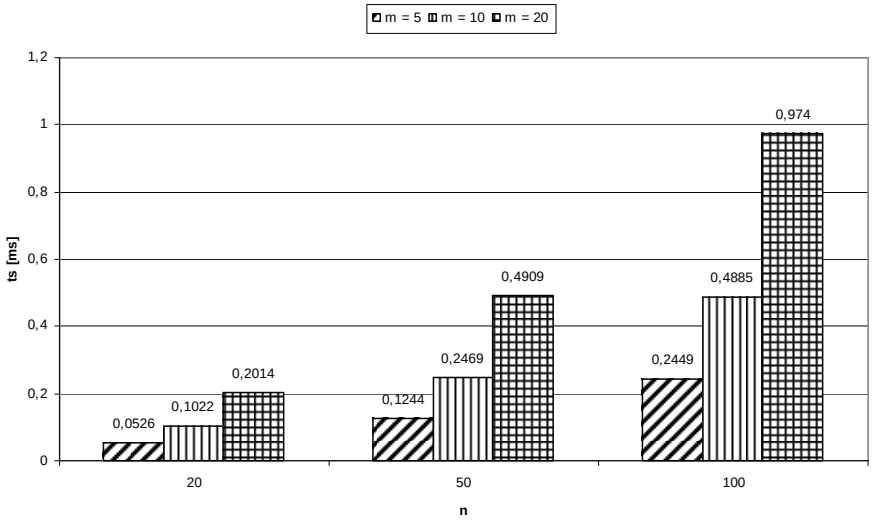
Algorytm równoległego wyznaczania funkcji celu dla permutacyjnego problemu przepływowego został zaimplementowany w języku C (CUDA) dla GPU i uruchomiony na 128-procesorowej karcie GPU Tesla C870 firmy nVidia dysponującej mocą obliczeniową 512 GFLOPS. Eksperymenty obliczeniowe zostały przeprowadzone dla instancji testowych Taillarda [12]. Zestaw instancji Taillarda składa się z 120 przykładów podzielonych na grup, o różnych rozmiarach. Dla każdego rozmiaru *n x m*: *20 x 5*, *20 x 10*, *20 x 20*, *50 x 5*, *50 x 10*, *50 x 20*, *100 x 5*, *100 x 10*, *100 x 20*, *200 x 10*, *200 x 20*, *500 x 20* występuje 10 instancji. Liczba procesorów GPU użytych do wyznaczenia wartości funkcji celu przez algorytm równoległy jest równa liczbie maszyn (*m*).

W celu wyznaczenia przyspieszenia, jakie można uzyskać wykorzystując *m* procesorów GPU do wyznaczenia wartości funkcji celu została zaimplementowana sekwencyjna wersja algorytmu uruchamiana dla jednego procesora GPU. W Tabeli 1 zostały zamieszczone wyniki eksperymentów obliczeniowych w postaci czasów trwania obliczeń dla algorytmu sekwencyjnego i równoległego oraz przyspieszenia obliczeń. Wartość przyspieszenia względnego *s* została wyznaczona z zależności $s = \frac{t_s}{t_p}$, gdzie t_s oznacza czas obliczeń dla algorytmu sekwencyjnego, a t_p czas obliczeń dla algorytmu równoległego. Rys. 4. zawiera zestawienie czasów obliczeń funkcji celu dla algorytmu sekwencyjnego.

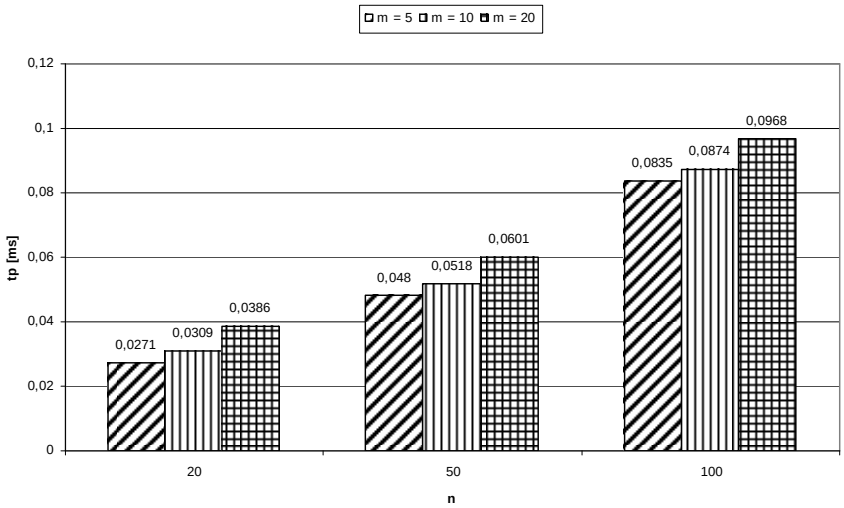
Tab. 2. Wyniki przeprowadzonych eksperymentów obliczeniowych dla przykładów testowych Taillarda (oraz dodatkowo wygenerowanego przykładu z 50 maszynami).

$n \times m$	p	t_p [ms]	t_s [ms]	S
20 × 5	5	0,0271	0,0526	1,94
20 × 10	10	0,0309	0,1022	3,31
20 × 20	20	0,0386	0,2014	5,22
50 × 5	5	0,0480	0,1244	2,59
50 × 10	10	0,0518	0,2469	4,76
50 × 20	20	0,0601	0,4909	8,16
100 × 5	5	0,0835	0,2449	2,93
100 × 10	10	0,0874	0,4885	5,59
100 × 20	20	0,0968	0,9740	10,06
200 × 10	10	0,1582	0,9716	6,14
200 × 20	20	0,1697	1,9403	11,43
500 × 50	50	0,3919	4,8392	12,35

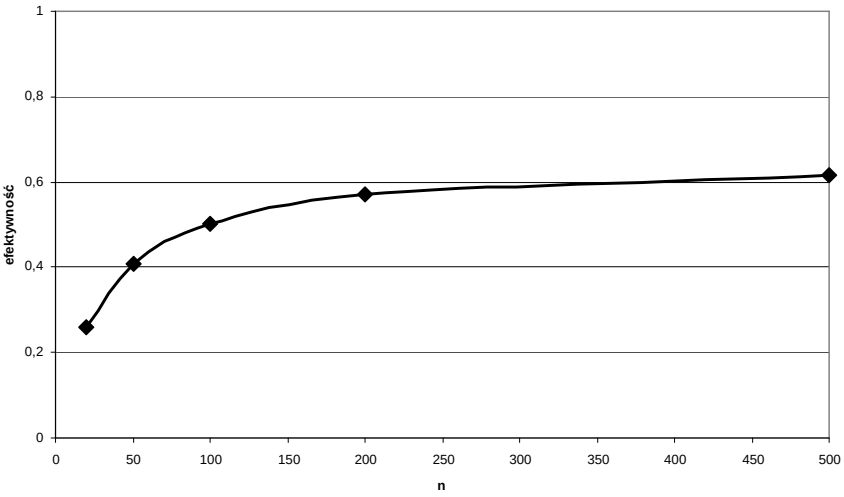
Można zauważyć, że wzrost rozmiaru problemu powoduje wzrost czasu działania algorytmu. Dla ustalonej liczby zadań dwukrotne zwiększenie liczby maszyn powoduje dwukrotny wzrost czasu obliczeń. Rys. 5. zawiera zestawienie czasów obliczeń funkcji celu dla algorytmu równoległego. Przy ustalonej liczbie zadań zwiększenie liczby maszyn powoduje niewielki wzrost czasu obliczeń. Wzrost rozmiaru problemu zwiększa osiągane przyspieszenia uzyskanego poprzez algorytm równoległy w stosunku do algorytmu sekwencyjnego.
Rys. 6. potwierdza teoretyczną efektywność wyznaczoną na podstawie Tw. 1. I zaprezentowaną na Rys. 3. Rys. 7. ilustruje zależność przyspieszenia od liczby procesorów użytych do obliczeń. Uzyskane wyniki eksperymentalne w pełni potwierdzają analizę teoretyczną.



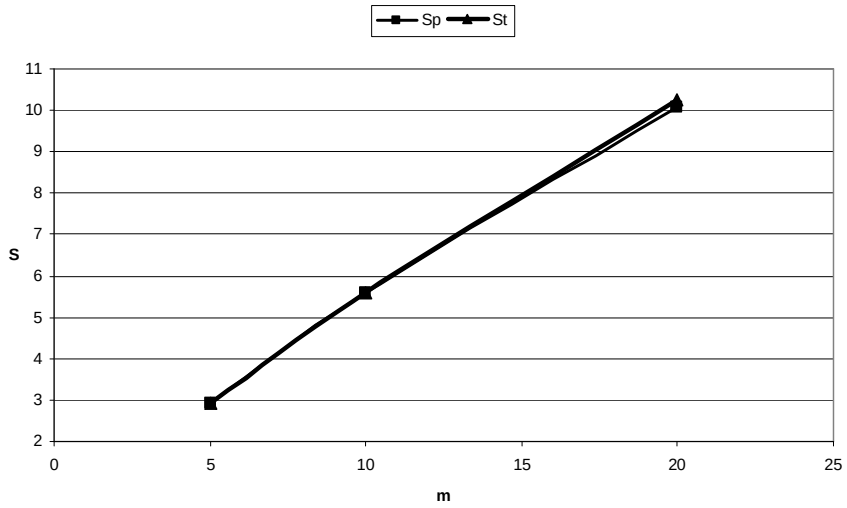
Rys. 4. Zestawienie czasów działania algorytmu sekwencyjnego.



Rys. 5. Zestawienie czasów działania algorytmu równoległego. Zaobserwować można znacznie mniejszą zależność czasu działania algorytmu od liczby maszyn *m* w stosunku do algorytmu sekwencyjnego.



Rys. 6. Wyznaczona eksperymentalnie **e**fektywność metody z dowodu Tw. 1., potwierdzająca wyniki teoretyczne zilustrowane na Rys. 3.



Rys. 7. Teoretyczne i wyznaczone eksperymentalnie przyspieszenie dla *n=100*. Uzyskane przyspieszenie jest prawie-liniowe (*almost-linear speedup*).

5. Wnioski

W pracy proponujemy wykorzystanie nowoczesnych procesorów obliczeniowych GPU w celu przyspieszenie działania algorytmów poszukiwań lokalnych przy rozwiązywaniu problemu przepływowego. Uzyskane wyniki pozwalają uzyskać przyspieszenie obliczeń proporcjonalne do liczby maszyn *m* występującej w problemie. Wyniki zaprezentowane w niniejszej pracy można rozszerzyć na model EREW PRAM z wyłącznością odczytu, co wymaga dodatkowego narzutu czasowego $O(\log n)$, choć architektura procesora obliczeń GPU pozwoliła na wykorzystanie implementacji z możliwością równoległego odczytu CREW. Z drugiej strony, wykorzystanie pamięci współdzielonej ze swobodnym dostępem obarczone jest ogromnym opóźnieniem czasowym (400-600 cykli procesora) w stosunku do czasu dostępu do pamięci lokalnej procesora lub też pamięci stałej czy pamięci tekstur. Dlatego też dalsze przyspieszenie algorytmu jest możliwe pod warunkiem dostosowania stosowanych metod do specyfiki dostępu do pamięci procesora obliczeń GPU.

Dodatek A. Specyfikacja techniczna procesora obliczeń GPU nVidia Tesla C870

- całkowity rozmiar pamięci globalnej 1,61 GB
- liczba multiprocesorów 16
- liczba rdzeni (procesorów) 128
- całkowity rozmiar pamięci stałej 65536 KB
- całkowity rozmiar pamięci współdzielonej przypadającej na jeden blok 16384 KB
- liczba rejestrów dostępna dla każdego bloku 8192
- częstotliwość zegara 1,35 GHz

Literatura

1. Aarts E.H.L., Verhoeven M., Local search, in Annotated bibliographies in Combinatorial Optimization (Dell'Amico M., Maffioli F., Martello S., eds.), 163-180, Wiley, 1997.
2. Adleman L., Molecular computation of solutions to combinatorial problems. Science (266) 1994, 1021-1024.
3. Bożejko W., Równoległy algorytm dynasearch dla jednomaszynowego problemu szeregowania zadań, Systemy sterowania (red. W.Greblicki, Cz. Smutnicki), WKŁ Warszawa 2005.
4. Bożejko W., J. Pempera, A. Smutnicki, Parallel single-thread strategies in scheduling, Lecture Notes in Artificial Intelligence 5097, Springer, 2008, 995-1006.
5. Bożejko W., J. Pempera, A. Smutnicki, Multi-thread parallel metaheuristics for the flow shop problem, in: International Conference on Artificial Intelligence and Soft Computing, IEEE Computational Intelligence Society - Poland Chapter and the Polish Neural Network Society, editors: L. Zadeh, L. Rutkowski, R. Tadeusiewicz, J.Zurada, 2008, 454-462.
6. Bożejko W., Wodecki M., Parallel path-relinking method for the flow shop scheduling problem, Lecture Notes in Computer Science 5101, Springer, 2008, 264-273.
7. Cormen T.H., Leiserson C.E., Rivest R. L., Wprowadzenie do algorytmów, WNT Warszawa 1997
8. Fiechter C.N., A parallel tabu search algorithm for large traveling salesman problems, Discrete Applied Mathematics 51 (1994), 243-267.
9. Nowicki E., Smutnicki C., A fast tabu search algorithm for the permutation flow-shop problem, European Journal of Operational Research 91, (1996), 160-175.
10. Porto S.C., Ribeiro C.C., Parallel tabu search message-passing synchronous strategies for task scheduling under precedence constraints, Journal of Heuristics 1 (1995), 207-223.
11. Porto S.C., Ribeiro C.C., A tabu search approach to task scheduling on heterogeneous processors under precedence constraints, International Journal of High Speed Computing 7 (1995), 45-71.
12. Taillard E., Benchmarks for basic scheduling problems, European Journal of Operational Research 64, (1993), 278-285.

Dr Wojciech BOŻEJKO
Instytut Informatyki, Automatyki
i Robotyki
Politechniki Wrocławskiej
50-372 Wrocław, ul. Janiszewskiego 11-17
e-mail: wojciech.bozejko@pwr.wroc.pl

Mgr inż. Mariusz UCHROŃSKI
Instytut Informatyki, Automatyki
i Robotyki
Politechniki Wrocławskiej
50-372 Wrocław, ul. Janiszewskiego 11-17
e-mail: mariusz.uchronski@pwr.wroc.pl