Optymalizacja, TSP,
Obliczenia równoległe,UPC,
Algorytmy genetyczne

SCHUBRING Tadeusz [1]

OBLICZENIA RÓWNOLEGŁE W ZADANIACH TRANSPORTOWYCH

W artykule przedstawiono koncepcję zastosowania architektur równoległych systemów komputerowych do rozwiązywania zadań transportowych na przykładzie problemu TSP (Traveling Salesman Problem). Uwzględniono możliwości wykorzystania paradygmatu programowania równoległego i środowiska programistycznego Berkeley UPC oraz metod genetycznych rozwiązywania zadań TSP.

PARALLEL CALCULATIONS IN TRANSPORT PROBLEMS

In the article concepts were presented of using architectures of parallel computer systems for solving transport problems on the example of TSP (Traveling Salesman Problem). Possibilities of using the paradigm for parallel programming including Berkeley UPC programming environment and genetic methods for solving TSP problems.

1. WSTĘP

Istotnym czynnikiem ułatwiającym podejmowanie decyzji w wielu dziedzinach m.in. w logistyce są wyniki analiz wspomaganych obliczeniami komputerowymi wykonywanymi na podstawie przyjętych modeli wynikających z zaawansowanych analiz świata rzeczywistego. Rozwój metod programowania oraz sprzętu komputerowego tworzy warunki do wzrostu potencjału obliczeniowego, a więc i zwiększenia możliwości wykorzystania wspomagania komputerowego podejmowania decyzji. Do klasycznych problemów optymalizacyjnych pomagających w podejmowaniu decyzji jest niewątpliwie tradycyjne zadanie komiwojażera czyli TSP (*Traveling Salesman Problem)* .

2. RÓWNOLEGŁE ROZWIĄZYWANIE PROBLEMU TSP

2.1 Złożoność problemu TSP

Koncepcyjnie sformułowanie zadania jest proste – komiwojażer musi odwiedzić dokładnie jeden raz każde z m miast należących do zbioru V oraz i wrócić do miasta startowego tak, aby suma wag krawędzi (najczęściej reprezentujących odległości między miastami) była najmniejsza. Inaczej mówiąc sprowadza się to, w ujęciu sieciowym, do znalezienia cyklu Hamiltona (w logistyce – marszruty) o minimalnej sumie wag [1,2,5].

[1]. Politechnika Radomska, Wydział Mechaniczny; 26-600 Radom; ul. Krasickiego 54.
tel: + 48 48 361-71-22, e-mail: t.schubring@pr.radom.pl, tchu@tlen.pl

Liczba dopuszczalnych rozwiązań zadania symetrycznego TSP może sięgać (m-1)!/2. Poszukiwanie rozwiązania Problemu poprzez przeglądanie i porównywanie marszrut jest klasy złożoności O(m!). Przy 50 miastach otrzymujemy ok. $3x10^{62}$ wariantów. Metoda „brutalna" bezpośredniego przeglądu i wyboru najlepszego rozwiązania dla takiej lub większej liczby miast nie naddaje się do praktycznej realizacji. Z tych powodów stosowane są metody przybliżone, których złożoność czasowa jest klasy O(W(m)), gdzie W jest wielomianem ilości miast. Algorytmy TSP znajdują zastosowanie nie tylko w logistyce, ale także w bioinformatyce, sieciach komputerowych, telefonii komórkowej i wielu innych dziedzinach, gdzie istnieją potrzeby wyznaczania racjonalnych tras.

2.2 Algorytm genetyczny rozwiązywania zadania zadania TSP

Do przybliżonego rozwiązywania problemów optymalizacyjnych, w tym zadania komiwojażera, stosowanych jest wiele algorytmów , a w szczególności algorytm zachłanny Dijkstry, algorytm symulowanego wyżarzania, algorytm Belmana-Forda, algorytmy genetyczne i inne. Algorytmy genetyczne [3,4] są konstruowane na podstawie analizy mechanizmów doboru naturalnego i dziedziczności obserwowanych w przyrodzie. Algorytmy te prowadzą do osiągnięcia przybliżonego rozwiązania optymalnego. Należą one do algorytmów uniwersalnych pozwalających na rozwiązywanie dużych zadań o względnie korzystnej złożoności obliczeniowej.

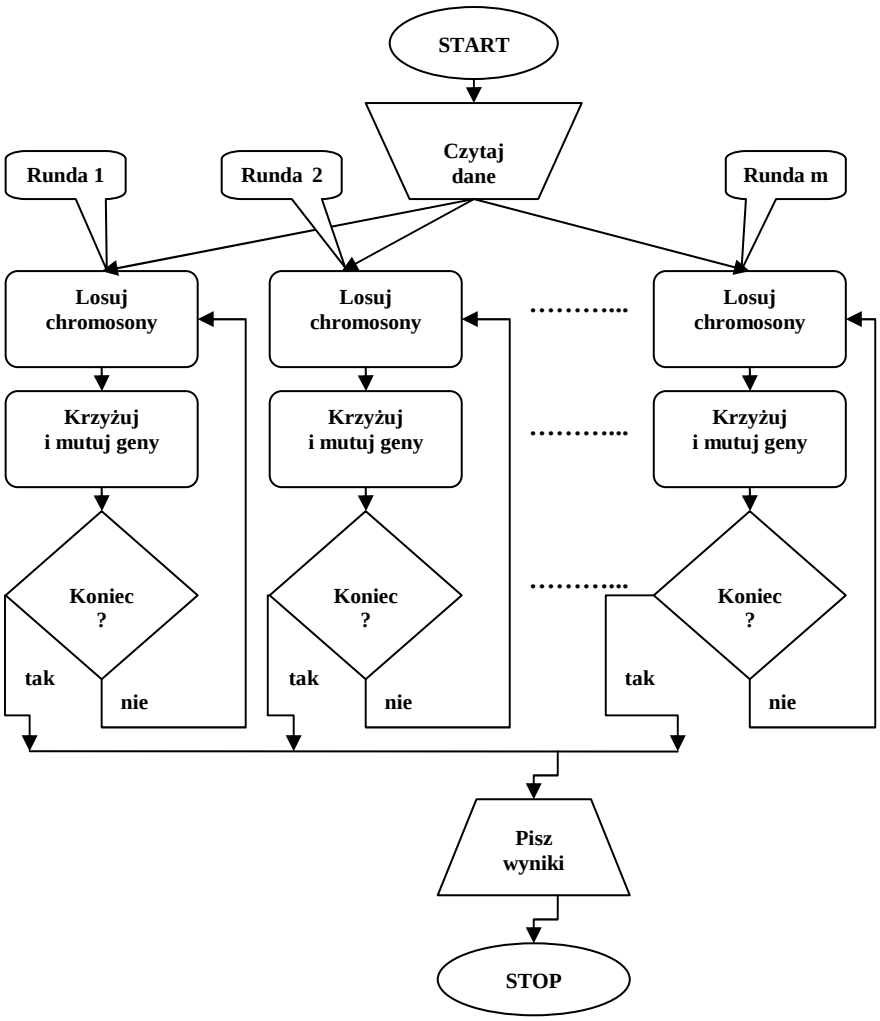
W wielu algorytmach (w tym genetycznych) do rozwiązywania zadania TSP możliwe jest zastosowanie mechanizmów programowania równoległego. Zrównoleglanie algorytmów może być prowadzone nie tylko w fazach powtarzania algorytmów dla różnych parametrów sterujących, ale również w operacjach cząstkowych typu kopiowanie danych, sortowanie, obliczanie sum, statystyk, iloczynów, czy generowanie liczb pseudolosowych. Do implementacji zrównoleglenia mogą być używane takie mechanizmy, jak: programowanie wielowątkowe w środowiskach programistycznych (Java, C++, Delphi, MS Visual Studio i inne), biblioteki MPI, czy też wyspecjalizowane języki do obliczeń równoległych. Przykładem takiego współczesnego języka do obliczeń równoległych jest Berkeley UPC (Unified Parrallel C) używający modelu SPMD (Single Program Multiple Data) [5]. W szczególności mogą być zrównoleglane algorytmy genetyczne, co prowadzi do redukowania czasu rozwiązywania zadań TSP.

Algorytm genetyczny rozwiązywania zadania TSP można zrealizować w następujących etapach:

1. Wygenerowanie pseudolosowej populacji początkowej chromsomów czyli łańcuchów, z których każda reprezentuje trasę (dopuszczalne rozwiązanie zadania TSP);
2. Powtórzenie operacji polegających na tworzeniu populacji potomków z populacji poprzedniej iteracji na podstawie funkcji kryterium zaleznej od sumy wag, metodami krzyżowania lub mutowania osobników.
3. Zatrzywanie algorytmu na podstawie oceny dokładności, czasu lub liczby wykonanych iteracji.

2.3 Równoległa implementacja algorytmu genetycznego dla TSP

W opisanej metodzie genetycznej istotne jest nie tylko wykonywanie iteracji prowadzących do poprawiania tras dla wylosowanej początkowej populacji (która dalej jest modyfikowana), ale również powtarzanie całego algorytmu począwszy od wylosowania populacji początkowej. Powtórzenie całego algorytmu będziemy dalej nazywali rundą. Wprowadzie danych i wyprowadzenie wyników odbywa się tylko raz odpowiednio na początku i na końcu algorytmu. Powtarzanie poszczególnych rund może odbywać się w ramach współbieżnych wątków.



Rys.1. Schemat algorytmu genetycznego wykonywanego we współbieżnych rundach

Każdy proces musi mieć do dyspozycji lokane struktury danych potrzebne do realizacji jednej rundy takie, jak tablice genów (do przetwarzania tras w postaci chromosonów) i tablice ocen kryteriów będących podstawą do krzyżowania i mutowania osobników. Rundy będą też korzystały ze wspólnych obszarów danych wejściowych takich, jak zmienne globalne i tabela zawierająca wagi krawędzi łączących węzły. Potrzebna jest też wspólna tablica do zapamiętywania tras wyznaczonych w ramach poszczególnych rund.

2.3 Równoległy program do rozwiązywania TSP w środowisku Berkeley UPC

Do imlementacji tak sformuowanego algorytmu doskonale naddaje się język Berkeley UPC. Język UPC jest rozszerzeniem języka C dedykowanym do tworzenia dużych wydajnych systemów obliczeniowych. UPC rozszerza standard ISO C 99 w zakresie możliwości tworzenia jawnego modelu równoległości, z wykorzystaniem prywatnej, wspólnej dzielonej i rozproszonej pamięci oraz mechanizmów synchronizacji dostępu do wspólnych zasobów. Język UPC został opracowany z wykorzystaniem doświadczeń wcześniejszych rozszerzeń równoległych języka C, takich, jak Parallel C Preprocessor (PCP), AC, Split-C [6,7]. Darmową wersję (udostępnianą na licencji BSD) projektu UPC wraz z instrukcją instalacji w systemach klasy Unix/Linux, Windows, Mac OS X oraz dokumentację można pobrać ze strony internetowej http://upc.lbl.gov/ [6]. Dostępne są również komercyjne implementacje systemu takie HP UPC czy IBM UPC. Projekt UPC jest cały czas rozwijany i aktualizowany. W październiku 2011 pojawiła się najnowsza wersja Projektu. Do zespołu zajmującego się rozwojem projektu UPC należą m. in. Tarek El-Ghazawi, Christian Bell, Wei Chen, Jason Duell, Parry Husbands, Mike Welcome, Dan Bonachea , Rajesh Nishtala, Katherine Yelick , Paul Hargrove, Filip Blagojevic, Seung-Jai Min , Costin Iancu, Yili Zheng [6].

W środowisku Berkeley UPC uruchamiano program z wykorzystaniem statycznych wątków na komputerze PC z czterordzeniowym procesorem AMD PHENOM 2.2 GHz z 4GB RAM. Program w języku UPC kompilowano poprzez wydanie zlecenia w terminalu znakowym systemu Linux:

upcc –pthreads=n tsp.upc –o tsp.o,

gdzie n było liczbą watków statycznych, z którymi miała być uruchamiana aplikacja, tsp.upc – nazwą programu źródłowego w języku UPC, tsp.o –nazwą pliku wynikowego.

Uruchamianie aplikacji realizowano za pomocą polecenia upcrun:

upcrun tsp.o .

Jeżeli program w języku UPC jest uruchamiany w wielu wątkach (instancjach), to może on korzystać ze struktur danych, zarówno prywantych przypisanych poszczególnym watkom, jak i dzielonych (wspólnych) przez wszystkie wątki. Tworzenie dzielonych danych w języku UPC wymaga stosowania słowa kluczowego shared. Konwencje związane ze stosowaniem tego kwalifikatora, w powiązaniu z przydziałem zmiennych dla poszczególnych wątków zostały szczegółowo opisane w pozycji [7]. W tym przypadku ma sens użycie instrukcji pętli równoległej upc_forall. Instrukcja ta jest rozszerzeniem operacji for z języka C. Instrukcja upc_forall pozwala na opis globalnej pętli z przypisaniem

wątkom zadań wynikających z wykonywania poszczególnych iteracji. Ogólna postać tej instrukcji jest następująca:

upc_forall(wyr1; wyr2; wyr3; powiązanie)instrukcja; .

Wyrażenie powiązanie, wskazuje w ramach jakiego wątku będzie realizowana dana iteracja pętli. Wątki nie muszą być wykonywane zgodnie z kolejnymi krokami upc_forall wynikającymi z przeliczania wyrażeń wyr1; wyr2; wyr3, których znaczenie jest analogiczne, jak w instrukcji for. Skojarzenie parametru powiązanie z odpowiednim wątkiem może obejmować różne przypadki. Zostało to szczegółowo opisane w [7]. Dwie stałe dostępne w programach UPC odgrywają ważną rolę:

- THREADS – liczba wątków (instancji), z którą został uruchomiony program,
- MYTHREAD indeks (0 ÷ THREADS-1) aktualnie realizowanego wątku.

Szkielet programu w języku Berkeley UPC do przybliżonego rozwiązywania zadania TSP przedstawia się następująco:

```
#include <upc.h>
#include <upc_io.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
//definicje stałych pomocniczych
static shared double suma_min;
// wspólna dzielona zmienna - minimalna suma wag
 static shared int nrTrasy;
//wspólna dzielona zmienna nr nalpepszej znalezionej trasy
static shared[] double W[V+1][V+1];
// wspólna dzielona tablica wag
static shared[] int trasa[Lrund][V+1][2];
//wspólna dzielona tablica wyznaczonych tras
static shared[] double DLtrasy[Lrund];
//wspólna dzielona tablica długości wyznaczonych tras
//prototypy pomocniczych funkcji programu
 int main(void)
{
    //deklaracje zmiennych lokalnych wątków
  if(MYTHREAD==0)
    {
        //indentyfikacja czasu
        //wprowadzenie danych
     }
  upc_barrier;//bariera oczekiwania na zakończenie wątku zerowego
  upc_forall(mm=0;mm<Lrund;mm++;mm)
```

```
{
  //Powtarzanie rund w trybie wielowątkowym
  //Losowanie chromosonów
  //Krzyżowanie i mutacja
  DLtrasy[mm]=wyniki_suma;//zapamiętanie wyników rundy
  if(DLtrasy[mm]<suma_min)
        {suma_min=DLtrasy[mm];nrTrasy=mm;}
  //Wybór najlepszej dotyczasowej trasy
   upc_barrier;//bariera oczekiwania na zakończenie wątków rund
       if(MYTHREAD==0)
            {
                //identyfikacja czasu
                //wyprowadzenie wyników
            }
    upc_barrier;//bariera oczekiwania na zakończenie wątku zerowego
           return 0;
   }
   //implementacje  pomocniczych funkcji programu
```

2.3 Wyniki obliczeń

Obliczenia zostały przeprowadzone dla:

- 100 miast,
- współczynnika selekcji genów do krzyżowania na poziomie 20%
- wspólczynika mutacji 1%
- liczby iteracji w ramach jednej rundy równej 70
- liczby rund równej 500

Tab.1. Czasy wykonania programu TSP

Liczba statycznych watków	**Czas wykonania [s]**	**Długość trasy**
1	21	4016
2	12	4152
4	14	4046
6	11	3987
8	12	4035
10	16	4196
12	17	4090
14	21	4118

Czasy wykonywania programu dla różnych liczb statycznych wątków przedstawiono w Tab.1. Dla 2, 6 i 8 wątków na maszynie 4 rdzeniowej, w środowisku systemu Linux, obliczenia trwały najkrócej. Odchylenie wyników obliczeń minimalnych długości tras nie przekroczyły 3% w stosunku do wartości średniej i 5% względem minimalnej długości wyznaczonej trasy. Biorąc pod uwagę fakt, że system Brekeley UPC jest dedykowany do

tworzenia dużych wydajnych systemów obliczeniowych, można postawić tezę, że w systemie z większą liczbą rdzeni (węzłow liczących), zysk czasowy będzie jeszcze większy, przy wyborze racjonalnej liczby watków.

3. WNIOSKI

Język Berkeley UPC, dzięki wbudowanym mechanizmom dzielonej pamięci, gloalnej instukcji pętli równoległej i jawnemu modelowi równoległości, nie tylko pozwala na łatwą implementację algorytmów równoległych, ale także pokazuje kierunki zrównoleglania algorytmów sekwencyjnych. W równoległym systemie komputerowym racjonalny dobór liczby wątków w środowisku UPC gwarantuje przeprowadzenie efektywnych obliczeń. Przykładowe obliczenia przeprowadzono w środowisku UPC stosując algorytm genetyczny rozwiązywania zadania TSP. Berkeley UPC może być wykorzystywany nie na pojedynczych komputerach, ale również w rozbudowanych systemach zawierających wiele jednostek (rdzeni) liczących. Daje nam to możliwość łatwego przenoszenia kodów programów napisanych w UPC z systemów małej mocy do systemów wielkiej mocy i na odwrót. Znając potencjał środowiska UPC, można również zaplanować odpowiednią moc obliczeniową i przeprowadzić przetwarzanie korzystając z usług typu cloud computing (obliczenia w chmurze) w ramach modeli IAAS (Infrastructure as a Service) , PAAS (Platform as a Service) lub SAAS (Software as a service).

4. BIBLIOGRAFIA

[1] Banachowski L., Diks K. i inni: *Algorytmy i struktury danych*, Warszawa, PWN 2003.
[2] Jakubczyk K., *Wprowadzenie do algorytmów i struktury danych* , Radom, WPR 2005.
[3] Goldberg D.E., *Algorytmy genetyczne i ich zastosowania*, Warszawa, WNT 2003.
[4] Z. Michalewicz., *Algorytmy genetyczne + struktury danych = programy ewolucyjne*, Warszawa, WNT, 2003.
[5] Wróblewski P., *Algorytmy, struktury danych i techniki programowania*, Gliwice, Helion, 2003.
[6] *Berkeley UPC - Unified Parallel C*, http://upc.lbl.gov/, dnia 15.10.2011.
[7] El-Ghazawi T., Carlson W. i inni, *UPC – distrubuted shared memory programing*, New Jersey, John Wiley & Sons, 2005.