Toruń 27.01.2007

Programowanie
Równoległe i Rozproszone

Zespół nr 9 w składzie :

Przemysław Krysztowiak

Piotr Modzelewski

Paweł Olzacki

Bartłomiej Żuchowski

1.Abstrakt

Problem jaki nas będzie interesował, to **problem przecinania się zbioru odcinków**, w którym mamy za zadanie sprawdzić, czy w danym zbiorze odcinków S istnieją dwa odcinki, które się przecinają. Do konstrukcji algorytmu sprawdzającego, użyjemy techniki **zamiatania**.

2.Specyfikacja problemu

Dane:

Zbiór $S = \{ s_1, s_2, ..., s_n \}$ składający się z n odcinków w R^2. Koordynaty odcinka $s_i : i = 1, ..., n$ zadane są przez współrzędne jego końców $p_{iL}(x, y)$ oraz $p_{iR}(x, y)$. Punkty nie mogą się pokrywać, a także nie mogą być końcami odcinków wzajemnie prostopadłych.

Wynik:

Informacja czy doszło do przecięcia odcinków w przestrzeni danego procesu, poprzez zwrócenie wartości logicznej *TRUE*, jeśli jakaś para odcinków w S się przecina, *FALSE* zaś w przeciwnym przypadku. Zwracane są też współrzędne punktu przecięcia.

3.Algorytm równoległy

Stając przed zadaniem zaprojektowania algorytmu równoległego rozwiązującego nasz problem, musimy się skupić na:

- metodzie podziału naszego zadania,
- oraz wyborze odpowiedniej komunikacji.

Podział:

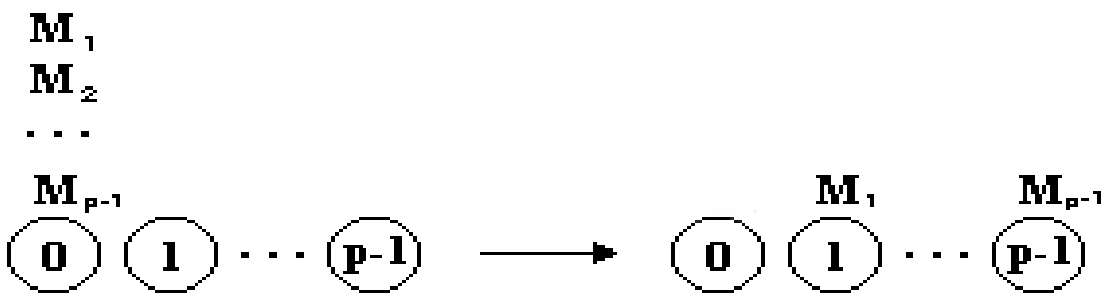
Płaszczyzna (prostokąt) na której znajdują się analizowane przez nas odcinki, zostaje podzielona (*by Master*) na m segmentów i rozdzielona

między taką samą ilość procesów (S*laves*).

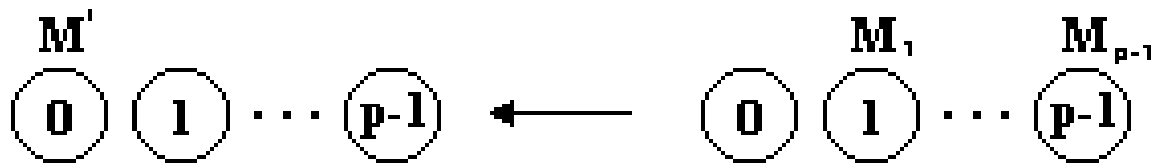
Do każdego procesu P_j, gdzie $j = 1, ..., m$ zostaje przekazany odpowiedni fragment płaszczyzny jego dotyczący. Stąd wymagana jest minimalna liczba procesów równa dwa, gdyż *Master* nie uczestniczy w obliczeniach algorytmu.

Komunikacja :

Na wstępie *Master* rozsyła do wszystkich procesów odpowiednie dane – operacja SCATTER.



Następnie przechodzi w stan nasłuchiwania w oczekiwaniu na wyniki – operacja GATHER.



Działanie algorytmu :

Będziemy przechowywać informację o porządku na odcinkach aktualnie przecinających miotłę w strukturze *T*, pozwalającej na wykonanie następujących operacji:

- INSERT(*T*, *s*) – wstawia odcinek *s* do porządku *T*,
- DELETE(*T*, *s*) – usuwa odcinek *s* z porządku *T*,
- ABOVE(*T*, *s*) – zwraca odcinek przed *s* w porządku *T*,
- BELOW(*T*, *s*) – zwraca odcinek po *s* w porządku *T*.

Przygotowanie danych :

1 $T \leftarrow \emptyset$

2 posortuj końce odcinków w *S* od lewej do prawej, rozstrzgając remisy na korzyść punktów o mniejszej współrzędnej *y*

Główna procedura wykonywana *by Slave*:

```
Line_Segment_Intersection(S)
for każdy punkt p na posortowanej liście końców odcinków
      do      if p jest lewym końcem ocinka s
              then INSERT(T, s)
                        if (ABOVE(T, s) istnieje i przecina s) lub (BELOW(T, s) istnieje i przecina s)
                        then return TRUE
              if p jest prawym końcem odcnika s
                  then if oba odcniki ABOVE(T, s) i BELOW(T, s) istnieją i ABOVE(T, s) przecina BELOW(T, s)
                        then return TRUE
                        DELETE(T, s)
return FALSE
```

4.Rząd złożności

Zlozonosc obliczeniowa dla N Odcinkow:

$$T_s = O((N) * \log(N))$$

$$T_p = T_{comp} + T_{comm} + T_{idle}$$

$$T_{comp} = O((\frac{(N+L)}{P}) * \log(\frac{(N+L)}{P})) + O((\frac{(N+L)}{P}) * \log(\frac{(N+L)}{P})),$$

gdzie L – liczba dodanych dla procesu pktow L < N, P – liczba procesow

$T_{comm} = O(N+L) + O(P)$ (scatter N+L punktów) i gather P punktów przecięcia)

$$T_p = O((\frac{(N+L)}{P}) * \log(\frac{(N+L)}{P})) + O(N+L) + O(P)$$

Przyspieszenie:

$$S = \frac{T_s}{T_p} = \frac{(O((N) * \log(N)))}{(O((\frac{(N+L)}{P}) * \log(\frac{(N+L)}{P})) + O(N+L) + O(P))}$$

Efektywność:

$$E=\frac{S}{P}\approx\frac{(O((\frac{N}{P})*\log(N)))}{(O((\frac{(N+L)}{P})*\log(\frac{(N+L)}{P}))+O(N+L)+O(P))}\approx O(\frac{(N*\log(N))}{((N+L)*\log(N+L))+P*(N+L)+P^2})$$

Koszt:

$$K_{par}=T_P*P=O(((N+L)+K*P)*\log(\frac{(N+L)}{P}))+O(P*(N+L))+O(P*K_P)$$