Politechnika Wrocławska

STEROWANIE PROCESAMI DYSKRETNYMI

Temat 2 - Analiza i implementacja algorytmów dla problemu szeregowania zadań na maszynach równoległych (Pm||Cmax)

Wydział i kierunek	W12N, Automatyka i Robotyka
Autor i numery albumów	Michał Markuzel 275417
Termin zajęć	Poniedziałek 13:15-14:55
Prowadzący	dr inż. Agnieszka Wielgus

1 Opis problemu

Problem harmonogramowania zadań polega na przydzieleniu zadań do zasobów w taki sposób, aby zoptymalizować określone kryterium. W naszym przypadku analizujemy problem szeregowania zadań na m równoległych, identycznych maszynach (Pm||Cmax).

Dany jest zbiór M = 1, ..., i, ..., m m równoległych, identycznych maszyn oraz zbiór J = 1, 2, ..., j, ..., N zadań. Każde zadanie j charakteryzuje się:

- p_j - czas przetwarzania (processing time) - czas potrzebny na wykonanie zadania

Dodatkowe założenia problemu:

- Każde zadanie może być wykonywane dokładnie przez jedną maszynę
- Maszyny są identyczne - czas wykonania danego zadania jest taki sam na każdej maszynie
- Zadania są niepodzielne - nie można przerwać wykonywania zadania i dokończyć go później

Celem optymalizacji jest minimalizacja czasu zakończenia wykonywania ostatniego zadania, czyli:

$$C_{max} = \max_{j \in J} C_j$$

gdzie C_j oznacza czas zakończenia zadania j.

1.1 Algorytm LSA (List Scheduling Algorithm)

1.1.1 Idea algorytmu

LSA (List Scheduling Algorithm) to prosty algorytm zachłanny dla problemu Pm||Cmax. Algorytm przydziela kolejne zadania do maszyny o aktualnie najmniejszym obciążeniu. Jest to implementacja strategii "jak najwcześniej"dla wielu maszyn równoległych.

1.1.2 Pseudokod

FUNKCJA LSAPlaning(zadania):
// Inicjalizacja obciążeń wszystkich maszyn
obciążenia_maszyn = wektor m zer
przydziały_zadań = wektor m pustych list

for każde zadanie z w zadania **do**
// Znajdź maszynę o najmniejszym obciążeniu
min_maszyna = indeks_minimum(obciążenia_maszyn)
// Przypisz zadanie do wybranej maszyny
dodaj z do przydziały_zadań[min_maszyna]
// Zaktualizuj obciążenie maszyny
obciążenia_maszyn[min_maszyna] += z.p
end for

ZWRÓĆ przydziały_zadań

1.2 Algorytm LPT (Longest Processing Time)

1.2.1 Idea algorytmu

LPT (Longest Processing Time) to rozszerzenie algorytmu LSA. Przed zastosowaniem strategii LSA, algorytm najpierw sortuje zadania malejąco według czasów wykonania. Dzięki temu dłuższe zadania są przydzielane w pierwszej kolejności, co często prowadzi do lepszego zrównoważenia obciążenia maszyn.

1.2.2 Pseudokod

FUNKCJA LPTPlaning(zadania):
// Sortowanie zadań według malejących czasów wykonania
posortowane_zadania = sortuj_malejąco(zadania, według=p)

// Zastosowanie algorytmu LSA do posortowanych zadań
ZWRÓĆ LSAPlaning(posortowane_zadania)

1.3 Algorytm przeglądu zupełnego (Brute Force)

1.3.1 Idea algorytmu

Algorytm przeglądu zupełnego dla P2||Cmax sprawdza wszystkie możliwe przydziały zadań do dwóch maszyn (2^n możliwości) i wybiera ten, który daje najmniejszy makespan. Dla każdego możliwego przydziału, obliczane są sumy czasów wykonania na obu maszynach, a makespan jest maksimum z tych sum. Choć algorytm gwarantuje znalezienie optymalnego rozwiązania, jego złożoność obliczeniowa ($O(2^n)$) ogranicza zastosowanie do małych instancji problemu.

1.3.2 Pseudokod

FUNKCJA BruteForcePlaning(zadania):
n = liczba_zadań
min_Cmax = NIESKOŃCZONOŚĆ
najlepszy_przydział = []

for mask = 0 **DO** $2^n - 1$ **do**
suma0 = 0, suma1 = 0
for i = 0 **DO** n-1 **do**
if mask & (1 « i) **then**
suma1 += zadania[i].p
else
suma0 += zadania[i].p
end if
end for
Cmax = max(suma0, suma1)
if Cmax < min_Cmax **then**
min_Cmax = Cmax
najlepszy_przydział = obecny_przydział
end if

end for **ZWRÓĆ** najlepszy_przydział

1.4 Algorytm programowania dynamicznego (Dynamic Programming)

1.4.1 Idea algorytmu

Algorytm programowania dynamicznego dla P2||Cmax traktuje problem jako zmodyfikowaną wersję problemu plecakowego. Celem jest znalezienie podzbioru zadań o sumie czasów wykonania najbliższej połowie całkowitego czasu wszystkich zadań. Algorytm wykorzystuje tablicę dp[j], która oznacza, czy można uzyskać sumę j z podzbioru rozważanych zadań. Złożoność czasowa to O(n·T), gdzie T to połowa sumy wszystkich czasów wykonania.

1.4.2 Pseudokod

FUNKCJA DynamicProgrammingPlaning(zadania):
    suma_całkowita = suma wszystkich czasów wykonania
    half = suma_całkowita / 2
    n = liczba_zadań

    dp[0..half] = [false, ..., false]
    parent[0..half] = [-1, ..., -1]
    dp[0] = true

for i = 0 **DO** n-1 **do**
  for j = half **DOWNTO** zadania[i].p **do**
    if dp[j - zadania[i].p] **AND NOT** dp[j] **then**
        dp[j] = true
        parent[j] = i
    end if
  end for
end for
    ZWRÓĆ odtwórz_rozwiązanie(dp, parent)

1.5 Algorytm PTAS (Polynomial-Time Approximation Scheme)

1.5.1 Idea algorytmu

PTAS dla P2||Cmax łączy dokładne rozwiązanie dla podzbioru dużych zadań z heurystycznym przydziałem pozostałych zadań. Dla zadanego parametru dokładności ε, algorytm wybiera k=1/ε największych zadań i sprawdza wszystkie możliwe ich przydziały (2^k kombinacji). Pozostałe zadania są przydzielane zachłannie do maszyny o mniejszym obciążeniu. Algorytm gwarantuje rozwiązanie o wartości nie większej niż (1+ε)·OPT.

1.5.2 Pseudokod

FUNKCJA PTASPlaning(zadania, epsilon):
    k = 1/epsilon
    posortuj zadania malejąco według czasów wykonania
    top_k = k największych zadań
    pozostałe = wszystkie zadania poza top_k
    min_makespan = NIESKOŃCZONOŚĆ

```
for mask = 0 DO 2^k − 1 do
        obciążenia = [0, 0]
        // Przydziel k największych zadań
    for i = 0 DO k-1 do
            maszyna = (mask & (1 « i)) ? 1 : 0
            obciążenia[maszyna] += top_k[i].p
    end for
        // Przydziel pozostałe zadania zachłannie
    for każde zadanie z w pozostałe do
            maszyna = (obciążenia[0]  obciążenia[1]) ? 0 : 1
            obciążenia[maszyna] += z.p
    end for
        aktualizuj_najlepsze_rozwiązanie()
end for    ZWRÓĆ najlepsze_rozwiązanie
```

1.6 Algorytm FPTAS (Fully Polynomial-Time Approximation Scheme)

1.6.1 Idea algorytmu

FPTAS dla P2||Cmax modyfikuje algorytm programowania dynamicznego poprzez skalowanie czasów wykonania zadań. Dla parametru ε, czasy są dzielone przez K=(ε·pmax)/(2n), gdzie pmax to najdłuższy czas wykonania. To redukuje zakres wartości w tablicy dp, zapewniając wielomianową złożoność czasową O(n^2/ε) przy gwarancji jakości rozwiązania (1+ε)·OPT.

1.6.2 Pseudokod

```
FUNKCJA FPTASPlaning(zadania, epsilon):
    n = liczba_zadań
    max_p = maksymalny czas wykonania
    K = (epsilon * max_p) / (2 * n)
    K = max(K, 1)

    // Przeskaluj czasy wykonania
for i = 0 DO n-1 do
        scaled_p[i] = [zadania[i].p / K]
end for

    suma = suma(scaled_p)
    half = suma / 2

    // Użyj programowania dynamicznego na przeskalowanych czasach
    dp[0..half] = [false, ..., false]
    dp[0] = true
for i = 0 DO n-1 do
  for j = half DOWNTO scaled_p[i] do
    if dp[j - scaled_p[i]] then
            dp[j] = true
            parent[j] = i
    end if
  end for
```

end for
ZWRÓĆ odtwórz_rozwiązanie(dp, parent)

2 Opis przeprowadzonego eksperymentu numerycznego

2.1 Środowisko testowe

Specyfikacja sprzętowa:

- **Procesor:** AMD Ryzen 7 7840HS with Radeon 780M Graphics
  - Liczba rdzeni: 8
  - Liczba wątków: 16
  - Częstotliwość bazowa: 400 MHz
  - Częstotliwość maksymalna: 5137 MHz
- **Cache:**
  - L1d: 256 KiB (8 instancji)
  - L1i: 256 KiB (8 instancji)
  - L2: 8 MiB (8 instancji)
  - L3: 16 MiB (1 instancja)

Środowisko programistyczne:

- **System operacyjny:** Ubuntu 22.04.5 LTS (Jammy Jellyfish)
- **Kompilator:** GCC 11.4.0
- **Język programowania:** C++17

2.2 Wartość kryterium, błąd względny (w %) oraz czas działania zaimplementowanych algorytmów.

Instancja	Zadania	LSA			LPT			DP			PTAS			FPTAS			Brute Force		
		Cmax	Błąd [%]	Czas [s]	Cmax	Błąd [%]	Czas [s]	Cmax	Błąd [%]	Czas [s]	Cmax	Błąd [%]	Czas [s]	Cmax	Błąd [%]	Czas [s]	Cmax	Błąd [%]	Czas [s]
instance_1_5	30	47	4.44%	2.0e-6	45	0%	2.0e-6	45	0%	4.0e-6	45	0%	249.0e-6	45	0%	3.0e-6	45	-	49.304
instance_1_15	30	129	3.2%	3.0e-6	125	0%	3.0e-6	125	0%	9.0e-6	125	0%	306.0e-6	125	0%	7.0e-6	125	-	49.612
instance_1_100	30	845	4.58%	2.0e-6	809	0.12%	2.0e-6	808	0%	32.0e-6	808	0%	271.0e-6	808	0%	30.0e-6	808	-	48.491

Tabela 1: Porównanie algorytmów dla różnych instancji problemu szeregowania zadań

3 Wnioski

3.1 Jakość rozwiązań

- **Algorytm Brute Force** gwarantuje znalezienie rozwiązania optymalnego, co potwierdza się w wynikach (Cmax = 808 dla instance_1_100, Cmax = 125 dla instance_1_15, Cmax = 45 dla instance_1_5).
- **Algorytmy aproksymacyjne (DP, PTAS, FPTAS)** osiągają takie same wyniki jak Brute Force, co świadczy o ich wysokiej skuteczności w znajdowaniu optymalnych lub bliskich optymalnym rozwiązań.
- **Algorytm LPT** (sortowanie malejąco według czasów wykonania) wypada nieznacznie gorzej od algorytmów aproksymacyjnych, z błędem względnym sięgającym maksymalnie 0.12% dla największej instancji.
- **Algorytm LSA** (w kolejności z pliku) konsekwentnie wypada najgorzej, z błędami względnymi od 3.2% do 4.58%.

3.2 Efektywność czasowa

- **Proste algorytmy (LSA, LPT)** są najszybsze, z czasami wykonania rzędu 2-3 mikrosekund dla wszystkich instancji.
- **Algorytmy aproksymacyjne (DP, FPTAS)** wykazują nieznacznie dłuższe czasy wykonania, ale wciąż pozostają bardzo efektywne (4-32 mikrosekund).
- **Algorytm PTAS** wymaga więcej czasu (249-306 mikrosekund), ale nadal jest bardzo szybki w porównaniu z Brute Force.
- **Brute Force** wykazuje zdecydowanie najdłuższe czasy wykonania (około 49 sekund), co czyni go niepraktycznym dla większych instancji.

3.3 Kompromis jakość-czas

- **DP i FPTAS** oferują najlepszy stosunek jakości rozwiązania do czasu wykonania, osiągając optymalne wyniki przy bardzo krótkim czasie obliczeń.
- **PTAS**, mimo dłuższego czasu wykonania, również znajduje rozwiązania optymalne, co czyni go dobrym wyborem gdy jakość jest ważniejsza niż czas.
- **LPT** stanowi rozsądny kompromis, oferując bardzo szybkie działanie przy zachowaniu dobrej jakości rozwiązań.
- **LSA**, mimo najkrótszego czasu wykonania, nie jest zalecany ze względu na znacznie gorszą jakość rozwiązań.