Szeregowanie zadań na dowolnej ilości równoległych procesorów

Autorzy:
Jacek Wieczorek (181043)
Tymon Tobolski (181037)

Prowadzący : mgr inż. Karolina Mokrzysz
Poniedziałek TP 11.00 - 13.00

Wydział Elektroniki
III rok

11 grudnia 2011

1 Opis problemu

Celem laboratorium jest zaimplementowanie algorytmu szeregowania n zadań na m równolegle działających procesorach $P_m||C_{max}$.

2 Symbole i oznaczenia

- m - liczba porcesorów
- n - liczba zadań
- C_{max} - przybliżony, maksymalny czas całkowity
- C - czas optymalny
- v - wektor współrzędnych m-wymiarowych
- v_i - współrzędna w i-tym wymiarze
- v_{min} - minimalna, największa współrzędna
- $t(z)$ - czas wykonania zadnaia z
- $p(i)$ - lista zadań i-tego procesora

3 Opis algorytmu

Pierwszy krok algorytmu opiera się na szybkiej, naiwnej metodzie LPT potrzebnej do wyznaczenia przybliżonego, maksymalnego czasu całkowitego (zawsze wiekszego lub równego od optymalnego). Aby rozważyć szeregowanie zadań na m procesorach wymagana jest m-wymiarowa struktura danych. Ze wzgledu na poziom skomplikowania implementacji takiej struktury, została ona uproszczona do postaci tablicy jednowymairowej o rozmiarze $(C_{max})^m$. W celu zamodelowania wielowymiarowości, współrzędne m-wymiarowe są przeliczane na indeks tablicy j i odwrotnie.

Wzór :

$$j = \sum_{i=1}^{m} v_i * m^{m-i}$$

Przykład :

$$\begin{aligned} m &= 4 \\ v &= (1,2,3,4) \\ j &= 1*4^3+2*4^2+3*4+4*1=112 \end{aligned}$$

Kolejnym krokiem jest ustawienie wartości -1 we wszystkich komórkach tabeli, oraz wartości 0 w pierwszej komórce (v = (0,0,...,0)). Dla każdego zadania $z \in \{1,...,n\}$ wyszukujemy kmórkę, której wartość jest równa numerowi poprzedniego zadania (dla $zadania1$ poszukujemy wartości 0). Następnie odnaleziony indeks tabeli jest zamienieniany na wektor współrzędnych v. W kolejnym kroku, dla każdego $k \in \{1,...,m\}$ obliczany jest nowy wektor współrzędnych v' na podstawie wzoru :

$$\begin{aligned} v' &= v \\ v'_k &= v'_k + t(z) \end{aligned}$$

Jeżeli v'_k jest mniejsze od C_{max} obliczany jest nowy indeks j' na podstawie wektora v' i w komórce tabeli o indeksie j' zapisywany jest numer zadania z.

Po rozdzieleniu wszystkich zadań wyszukiwane są te indeksy tabeli, które zawierają wartość n. Następnie wszystkie wyszukane indeksy zostają przekonwertowane na wektory współrzędnych. Spośród tych wektorów, wybierany jest ten, który posiada minimalną wartość największej współrzędnej. Ostatnim krokiem algorytmu jest odtworzenie ścieżki przydzielania zadań na poszczególne procesory. Począwszy od znalezionego zadania, aż do 0, algorytm działa według podanego schematu :

$$\begin{array}{l} \forall\ i \in \{1,...,m\} \\ \quad v'' = v_{min} \\ \quad v''_i = v''_i - z \\ \quad if\ tab[v'' \Rightarrow index] == z-1 \\ \quad\quad v_{min} = v'' \\ \quad\quad z = z-1 \\ \quad\quad p(i).add(z) \\ \quad break \end{array}$$

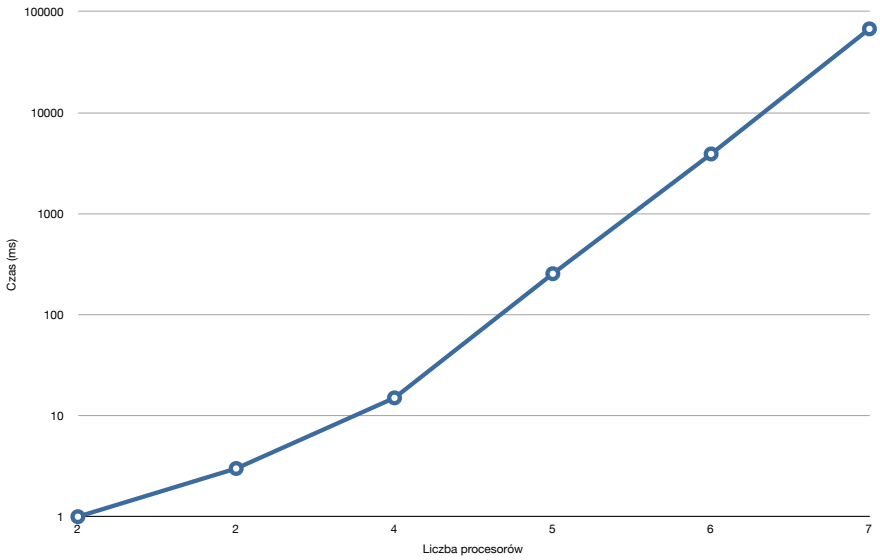
Podany algorytm powtarzaj póki $z > 0$

W momencie zakończenia algorytmu, każdy procesor ma przydzielony optymalny czas wykonywania zadań.

4 Analiza wyników

4.1 Stały zestaw zadań i zmienna ilość procesorów

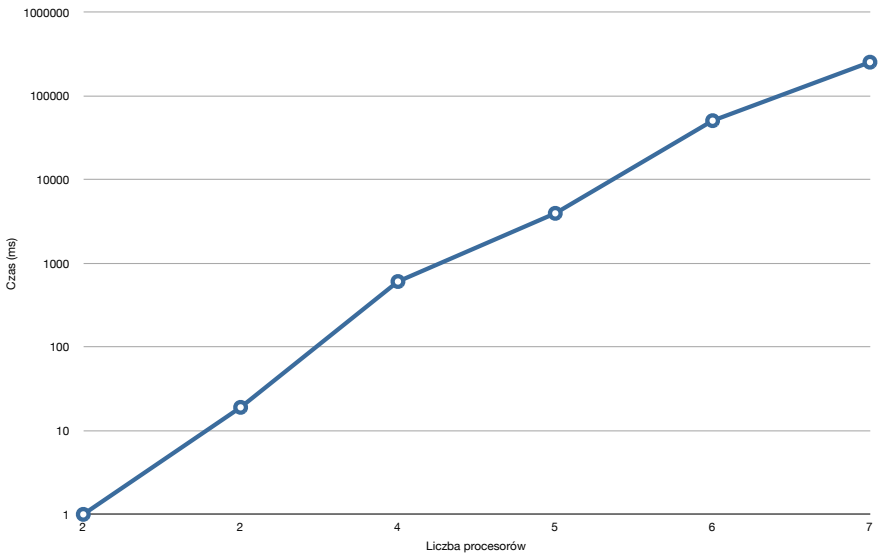
4.1.1 $m \in \{2, ..., 7\}$, $n = 5$, $range \in \{1, ..., 20\}$



Rysunek 1:

p	T
2	1
3	3
4	15
5	254
6	3889
7	67299

4.1.2 $m \in \{2, ..., 7\}$, $n = 10$, $range \in \{1, ..., 20\}$



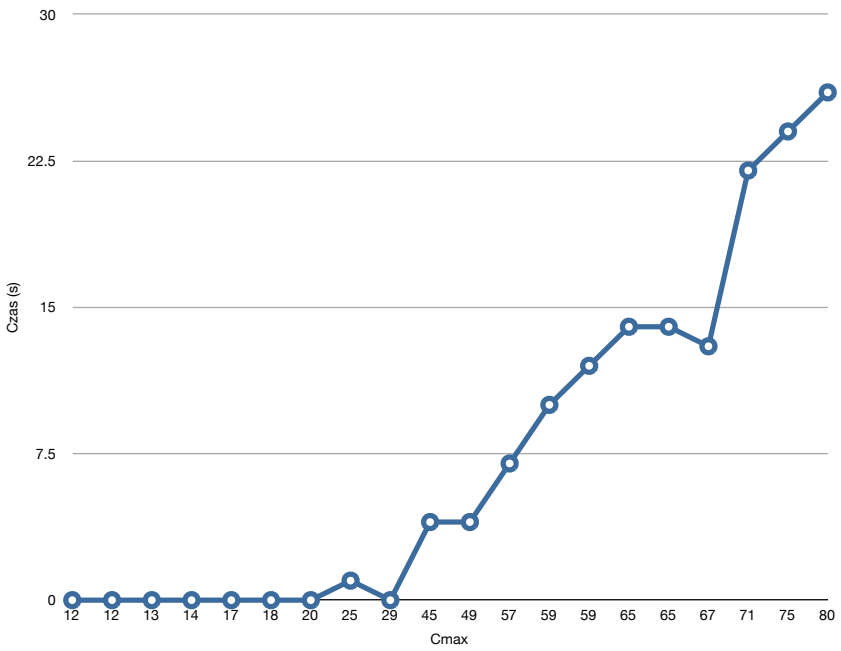
Rysunek 2:

p	T
2	1
3	19
4	606
5	3956
6	50652
7	253205

Jak widać na przestawionych wykresach (Rysunek 1, Rysunek 2) wraz ze wzrostem ilości procesorów, następuje wykladniczy wzrost czasu operacji szeregowania. Wieksza ilość zadań powoduje wzrost czasu operacji o kolejne rzędy wielkości.

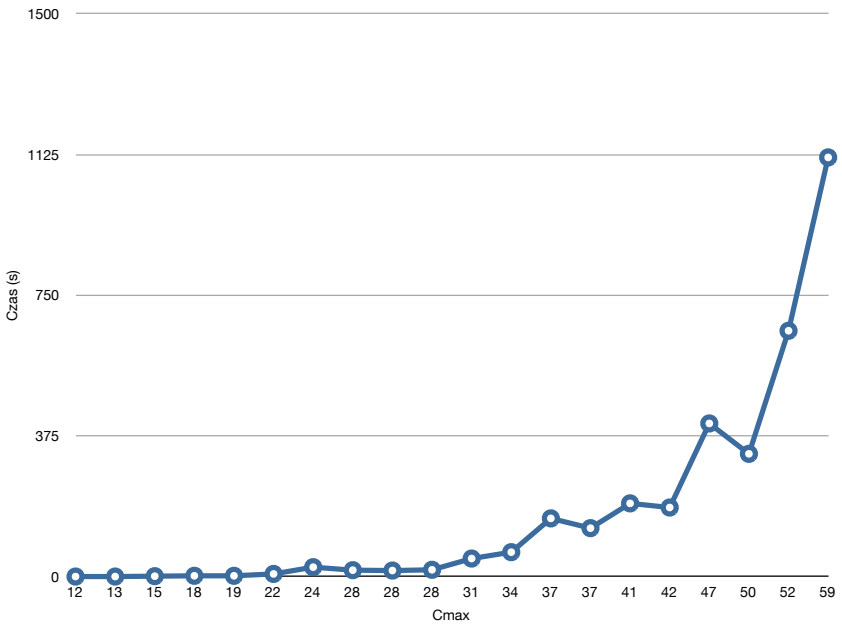
4.2 Zmienna wartość C_{max} i liczba zadań, stała ilość processorów

4.2.1 $m = 3$, $n \in \{1, ..., 20\}$, $range \in \{1, ..., 20\}$



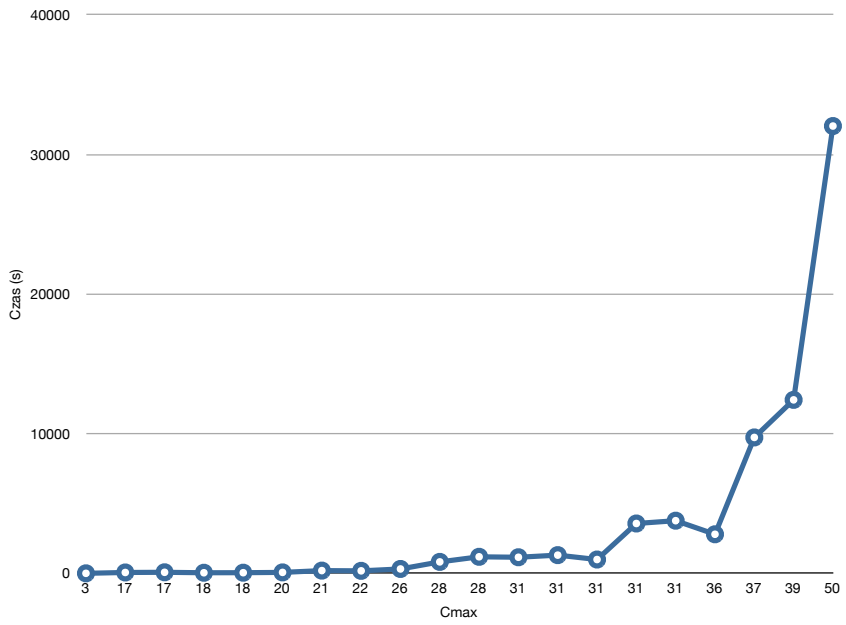
Rysunek 3:

4.2.2 $m = 4$, $n \in \{1, ..., 20\}$, $range \in \{1, ..., 20\}$



Rysunek 4:

4.2.3 $m = 5$, $n \in \{1, ..., 20\}$, $range \in \{1, ..., 20\}$



Rysunek 5:

Powyższe wykresy (Rysunek 3, Rysunek 4, Rysunek 5) obrazują wpływ wartości C_{max} na czas działania algorytmu. Dowodzi to wysokiej złożoności obliczeniowej algorytmu i wrażliwości na wielkość danych.

5 Wnioski

Zaimplementowany algorytm znajduje optymalne rozwiązanie zadanego problemu $NP-trudnego$, jednak wymaga relatywnie dużych zasobów (ilość dostępnej pamieci, szybkość procesora). Już dla małych wielkości algorytm wymagał większej ilości pamięci niż dostępna na testowym komputerze. Porównanie czasu wykonania z algorytmem naiwnym mija się z celem, ze względu na potrzebę uprzedniego obliczenia wartości C_{max} za pomocą algorytmu LPT, aby ustalić parametry wejściowe algorytmu dokładnego. Wyniki maksymalnego czasu wykonywania wszytskich zadań po uszeregowaniu przy długościach zadań nie wiekszych niż 20 (ograniczenia zasobów), nie rózniły się więcej niż o 3 jednostki. Nasuwa to wątpliwości co do potrzeby i opłacalności wykorzystania algorytmu dokładnego.

Mimo iż program posiada interfejs graficzny, jego testy przeprowadzone zostały za pomocą aplikacji konsolowej, by zminimalizować narzut czasowy GUI na jakość i wartość otrzymanych wyników.