Programowanie równoległe

Elementarne algorytmy

(Podstawa: Z. Czech. Wprowadzenie do obliczeń równoległych. PWN, 2010)

Andrzej Baran

baran@kft.umcs.lublin.pl

Charakterystyka ilościowa algorytmów

Przez algorytm równoległy (AR) rozumiemy pewną liczbą odrębnych algorytmów obliczeniowych, które mogą być wykonane jednocześnie.
Oceniając algorytm równoległy należy uwzględnić oprócz rozmiaru danych n dodatkowy parametr jakim jest liczba procesorów p.
Przy ocenie algorytmu bierze się pod uwagę takie jego cechy jak

- złożoność czasowa i pamięciowa
- przyspieszenie
- koszt algorytmu
- efektywność wykorzystania procesorów
- złożoność komunikacyjna
- przenośność

Charakterystyka ilościowa

Złożoność czasowa

Zakładamy, że algorytm równoległy (AR) rozwiązuje problem Z o rozmiarze n i korzysta z p procesorów. Złożoność czasowa (pesymistyczna) algorytmu jest

$$T(p,n) = \sup_{d \in D_n} \{t(p,d)\}$$

gdzie

- $t(p,d)$ = liczba kroków obliczeniowych wykonanych dla zestawu danych d
- p = liczba procesorów
- D_n = zbiór zestawów danych wejściowych o rozmiarz n

Charakterystyka ilościowa

Przyspieszenie AR

$$S(p,n) = \frac{T^*(1,n)}{T(p,n)}$$

Tutaj $T^*(1,n)$ jest złożonością czasową najlepszego algorytmu sekwencyjnego.
Ogólnie $T*(1,n) \leq T(1,n)$. Równość zachodzi tylko w przypadku gdy AR jest "kopią" algorytmu sekwencyjnego i nie ma komunikacji między procesorami.
Maksymalne przyspieszenie jest równe p, bo $S(p,n) \leq p$.
Na ogół $T(1,n) > T^*(1,n)$.
$S()$ jest miarą korzyści, którą osiąga się dzięki obliczeniom równoległym.

Złożoność $T(1,n)$ wyraża się przez złożoność części sekwencyjnej algorytmu $T^s(1,n)$ i równoległej $T^r(1,n)$. Przy tym

$$T(1,n) = T^s(1,n) + T^r(1,n)$$

Konieczność obliczeń sekwencyjnych przy realizacji AR wynika z zależności danych. Przykład:

$$x = a + b; \quad y = c * x; \qquad \text{– sekwencyjne}$$

$$x = a + b; \quad y = c * d; \qquad \text{– równoległe}$$

Przy założeniu, że $T^*(1,n)$ da się równomiernie rozdzielić między p procesorów (zazwyczaj niemożliwe), dostaniemy

$$S(p,n) = \frac{T^*(1,n)}{T(p,n)} \leq \frac{T^s(1,n) + T^r(1,n)}{T^s(1,n) + T^r(1,n)/p + T^o(p,n)}$$

gdzie $T^o(p,n)$ oznacza dodatkową złożoność wynikłą z organizacji obliczeń równoległych (komunikacja) oraz opóźnień.
$T^o(p,n)$ zazwyczaj rośnie wraz ze wzrostem p, ale $T^r(1,n)/p$ maleje i zachowanie $S(p,n)$ jest takie, że S najpierw rośnie z p a potem maleje (przy stałym rozmiarze problemu n).

Charakterystyka ilościowa

Koszt AR

Koszt AR

$$C(p,n) = pT(p,n)$$

Koszt jest liczbą “kroków” jaką wykonują wszystkie procesory, realizując AR.

Koszt organizacji obliczeń jest równy różnicy kosztu obliczeń i złożoności najlepszego AS (sekwencyjnego)

$$C^o(p,n) = pT(p,n) - T^*(1,n)$$

(= operacje nadmiarowe, puste, bezczynność procesorów, koszt odwołań do pamięci wspólnej)

Charakterystyka ilościowa

Efektywność wykorzystania procesorów

$$E(p,n) = \frac{S(p,n)}{p} = \frac{T^*(1,n)}{pT(p,n)}$$

Maksymalna wartość efektywności wynosi 1 (procesory pracują 100% czasu działania i koszty komunikacyjne są zerowe)
Algorytm jest optymalny pod względem kosztu jeśli spełniona jest równość

$$pT(p,n) = \Theta(T^*(1,n))$$

Efektywność AR optymalnego względem kosztu jest równa

$$E(p,n) = \Theta(1)$$

Charakterystyka ilościowa

Skalowalność

Skalowalność oznacza zwiększenie przyspieszeń równoległych lub utrzymanie stałej efektywności wraz ze wzrostem liczby procesorów.
Rozpatrzmy AR, który jest jakąś wersją AS. Mamy

$$E(p,n) = \frac{T(1,n)}{pT(p,n)}$$

Ponieważ w tym wypadku

$$C^o(p,n) = pT(p,n) - T^*(1,n) = pT(p,n) - T(1,n)$$

to

$$pT(p,n) = C^o(p,n) + T(1,n)$$

Dla efektywności dostaniemy

$$E(p,n) = \frac{T(1,n}{T(1,n) + C^o(p,n)} = \frac{1}{1 + \frac{C^o(p,n)}{T(1,n)}}$$

Ponieważ C^o rośnie z p i dla ustalonego rozmiaru n problemu złożoność $T(1,n) =$const, to widzimy stąd, że efektywność maleje wraz ze wzrostem liczby procesorów.

Z drugiej strony przy ustalonym p zwiększenie rozmiaru problemu n powoduje wzrost efektywności gdyż $T(1,n)$ rośnie z reguły szybciej niż $C^o(p,n)$.
Z wzoru widać, że dla utrzymania efektywności (zapewnienie skalowalności) systemu równoległego na stałym poziomie przy rosnącej liczbie procesorów należy zapewnić spełnienie związku

$$\frac{C^o(p,n)}{T(1,n)} = \Theta(1)$$

Ze wzrostem C^o powinien rosnąć koszt. Można to zapewnić zwiększając rozmiar problemu - należy rozwiązywać problemy większe! (Przykład)

Skalowalność

System równoległy jest skalowalny jeśli jego efektywność utrzymuje się na stałym poziomie poprzez jednoczesne zwiększanie liczby procesorów oraz zwiększanie rozmiaru problemu.

Prawo Amdahla (1967)

Rozważmy algorytm sekwencyjny o złożoności $T(1,n)$. Niech r oznacza część obliczeń równoległych, a s sekwencyjnych. Mamy wówczas

$$T^s(1,n) = sT(1,n), \quad T^r(1,n) = rT(1,n), s+r = 1.$$

Przyspieszenie po urównolegleniu

$$\begin{aligned} S(p,n) &\leq \frac{T^s(1,n)+T^r(1,n)}{T^s(1,n)+T^r(1,n)/p+T^o(p,n)} && (1)\\ &\leq \frac{sT(1,n)+rT(1,n)}{sT(1,n)+rT(1,n)/p} && (2)\\ &= \frac{1}{s+r/p} = \frac{1}{s+(1-s)/p} && (3) \end{aligned}$$

Wzór ten nosi nazwę prawa Amdahla i daje ograniczenie przyspieszenia będącego funkcją s oraz p.
Np. dla $p = 1000$, $s = 0.01$ (1%), $S \approx 90$; dla $p \to \infty$, $s = 0.01$, $S = 100$.

Minimalny element tablicy

```
// Plik: elminimalny.alg
// Dane: Tablica n elementowa, p procesorów, pamięć wspólna
// Zadanie: Znaleźć najmniejszy element tablicy. Obliczyć  złożoność,
//          przyspieszenie, koszt oraz efektywność algorytmu
=================================================================
 1 begin
 2   parfor P_i, 1 <= i <= n do
 3      b[i] = a[i\  // kopiowanie tablicy a do pomocniczej tablicy b
 4      k = n
 5   end parfor
 6   for (j=1; j<=log(n); j++)
 7      parfor P_i, 1<=i<=k/2 do
 8         if b[i] > b[i+k/2] then
 9            b[i] = b[i+k/2]
10         end if
11      end parfor
12      k=k/2;
13   end for
14   if i==1 then m=b[1] end if
15 end
```

Minimmalny element tablicy

Złożoność:

$$T(p,n) = T(n) = 1 + \log n = O(\log n)$$

Przyspieszenie:

$$S(n) = \frac{n-1}{1+\log n} = O(\frac{n}{\log n})$$

Koszt:

$$C(n) = n(1+\log n) = O(n \log n)$$

Efektywność:

$$E(n) = \frac{n-1}{n(1+\log n)} = O(\frac{1}{\log n})$$

Suma elementów tablicy $a[n]$; n procesorów

```
// Plik: suma.alg
// Dane: Tablica liczb a[n]; n dowolne
// Zadanie: Znaleźć sumę elementów tablicy a
-----------------------------------------------------------------
 1  begin
 2     parfor P_i, 1 <= i <= n do
 3        b[i] = a[i];  // kopiowanie tablicy a do pomocniczej tablicy b
 4        k = n;
 5     end parfor
 6     for (j=1; j<= ceil( log n ); j++)
 7        parfor P_i, 1<=i<= floor(k/2) do
 8           b[i] = b[i] + b[k+1-i];  // pierwszy+ostatni; drugi+przedostatni;
 9        end parfor
10      k=ceil(k/2);
11     end for
12     if i==1 then s=b[1] end if   // wynik
13  end
```

Złożoność, przyspieszenie, koszt i efektywność są podobne do tych w algorytmie wyszukiwania elementu najmniejszego.

Suma elementów tablicy; *p* procesorów

```
// Plik: suma-p.alg
// Dane: Tablica liczb a[n]; n dowolne
// Zadanie: Znaleźć sumę elementów tablicy a używając p procesorów
-----------------------------------------------------------------
 1  begin
 2    parfor P_i, 1 <= i <= p do
 3       g = \ceil(n/p) (i - 1) + 1
 4       b[i] = a[g];
 5       for (j=1; j<= ceil( n/p ) - 1; j++)  // sumowanie w segmantach
 6         if (g + j) <= n then  // ostatni segment może być niepełny
 7            b[i] = b[i] + a[g + j];
 8         end if;
 9       end for;
10    end parfor
11    Sumowanie b[1} + b[2] + ... b[p] w czasie O(log p)
12    za pomocą algorytmu poprzedniego
13  end
```

Suma elementów tablicy; p procesorów

Złożoność:

$$T(p,n) = c_1 \lceil n/p \rceil + c_2 \lceil \log n \rceil \approx c_1 n/p + c_2 \log p = O(n/p + \log p)$$

Przyspieszenie:

$$S(p,n) \approx \frac{c_1 n}{c_1 n/p + c_2 \log n} = O(\frac{n}{n/p + \log p})$$

Koszt:

$$C(p,n) \approx p(c_1 n/p + c_2 \log p) = O(n + p \log p)$$

Efektywność:

$$E(p,n) \approx \frac{c_1 n}{c_1 n + c_2 p \log p} = O(\frac{n}{n + p \log p})$$

Biorąc w powyższym zadaniu $p = n/\log n$ procesorów dostaniemy:
Złożoność:

$$T(p,n) \approx n/p + \log p < 2\log n = O(\log p)$$

Przyspieszenie:

$$S(p,n) \approx \frac{n}{n/p + \log n} \approx \frac{n}{2\log n} = O(\frac{n}{\log n})$$

Koszt:

$$C(p,n) = n + p\log p = 2n - \frac{n\log\log n}{\log n} < 2n = O(n)$$

Efektywność:

$$E(p,n) \approx \frac{n}{n + p\log p} = O(\frac{n}{2n - n\log\log n/\log n} = \Theta(1)$$

Suma elementów tablicy $a[n]$; n procesorów

Algorytm poszukiwania minimum i alg. sumowania (ostatni) mają złożoność $O(\log n)$, ale ten ostatni używa mniej, bo tylko $n/\log n$ procesorów. Do wyznaczenia sumy w czasie $O(\log n)$ potrzeba co najmniej $n/\log n$ procesorów. Koszt nie może bowiem przekroczyć złożoności najlepszego algorytmu sekwencyjnego. A więc,

$$C(p,n) = p\log n \geq T^*(1,n) \approx n,$$

a stąd

$$p \geq n/\log n.$$

Algorytm jest też optymalny względem kosztu bo dla $n \geq 4$, $r = 1$ oraz $s = 2$ mamy $r(n-1) < C(p,n)|_{p=n/\log n} \leq s(n-1)$ czyli $C(p,n) = Theta(T^*(1,n))$.
Złożoność $T(p,n)$ określają dwie składowe: n/p, która maleje z p oraz $\log p$, któryrośnie z p. Istnieje wobec tego pewna liczba procesorów p_{opt} dla której złożoność czasowa osiąga minimum:

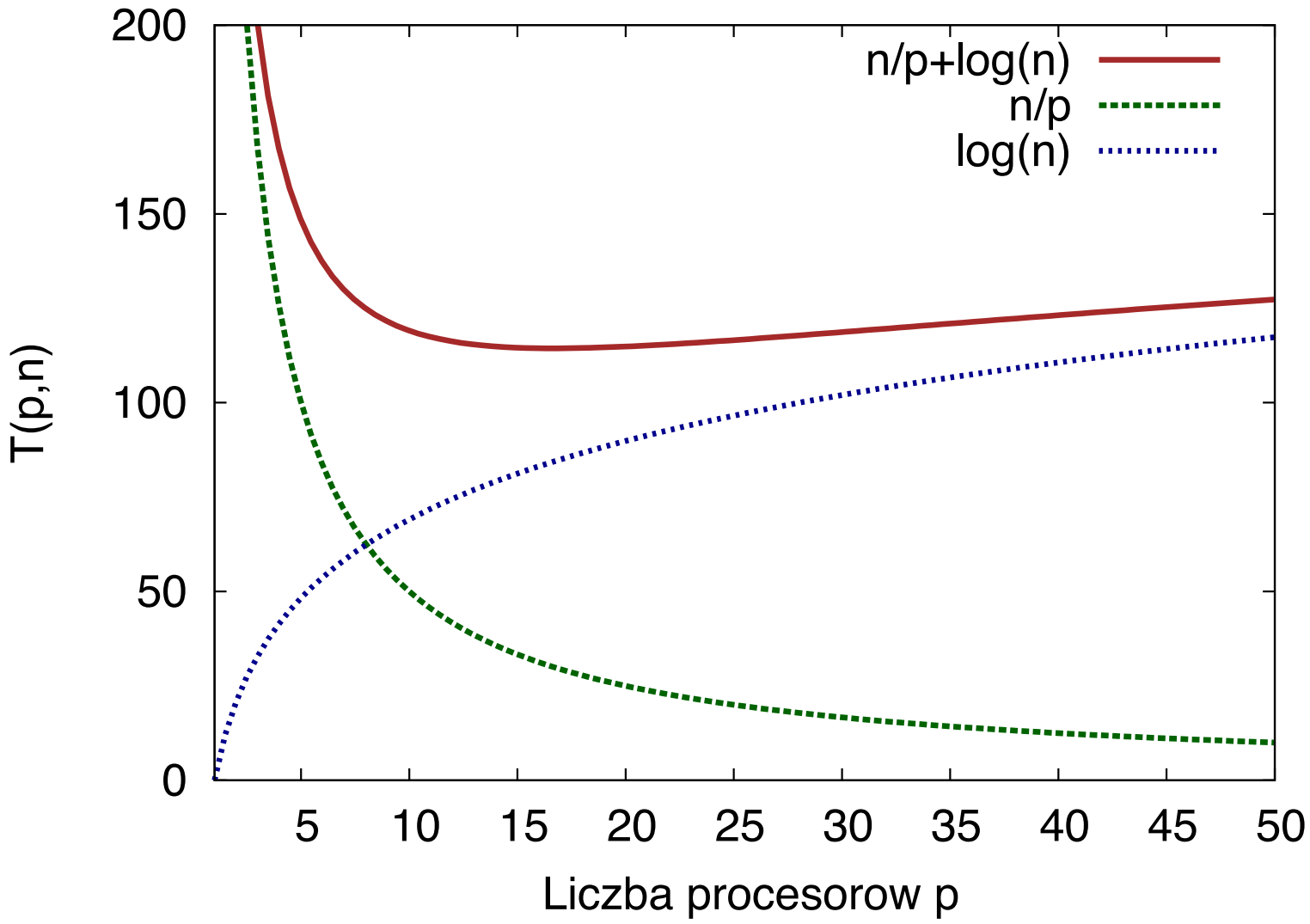
$$\frac{d}{dp}(c_1 n/p + c_2 \log p) = 0$$

prowadzi do $p_{opt} = c_1 n \ln 2/c_2$.

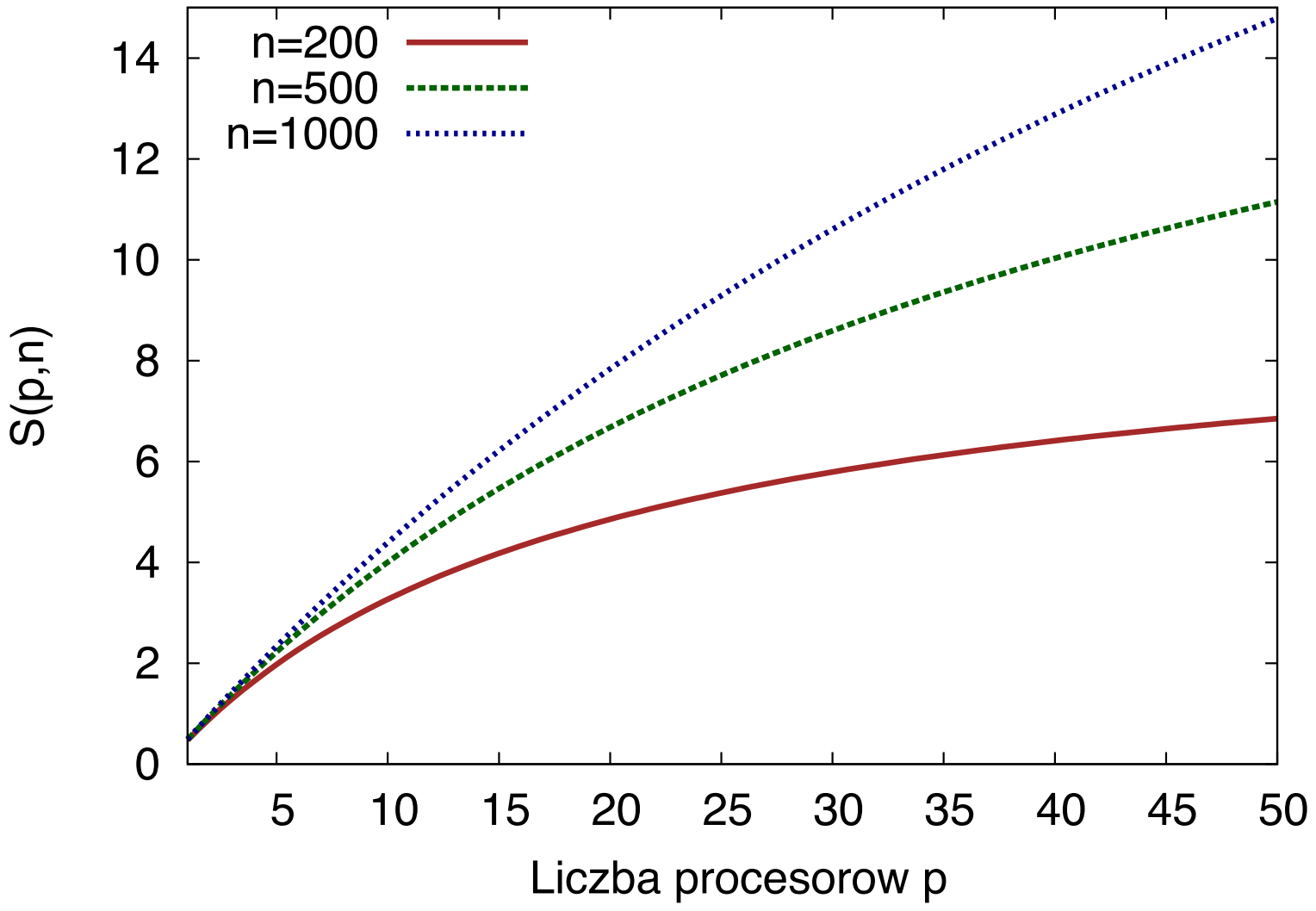
Redukcja

Problemy wyznaczania najmniejszego elementu lub sumy są szczególnymi przypadkami problemu redukcji: Dany jest zbiór $x_1, x_2, \ldots, x_n$. Należy obliczyć $x_1 \, op \, x_2 \, op \, x_3 \ldots op \, x_n$ gdzie op oznacza operację dwuargumentową łączna i przemienną (np. +, max. min, $\times$, itd).

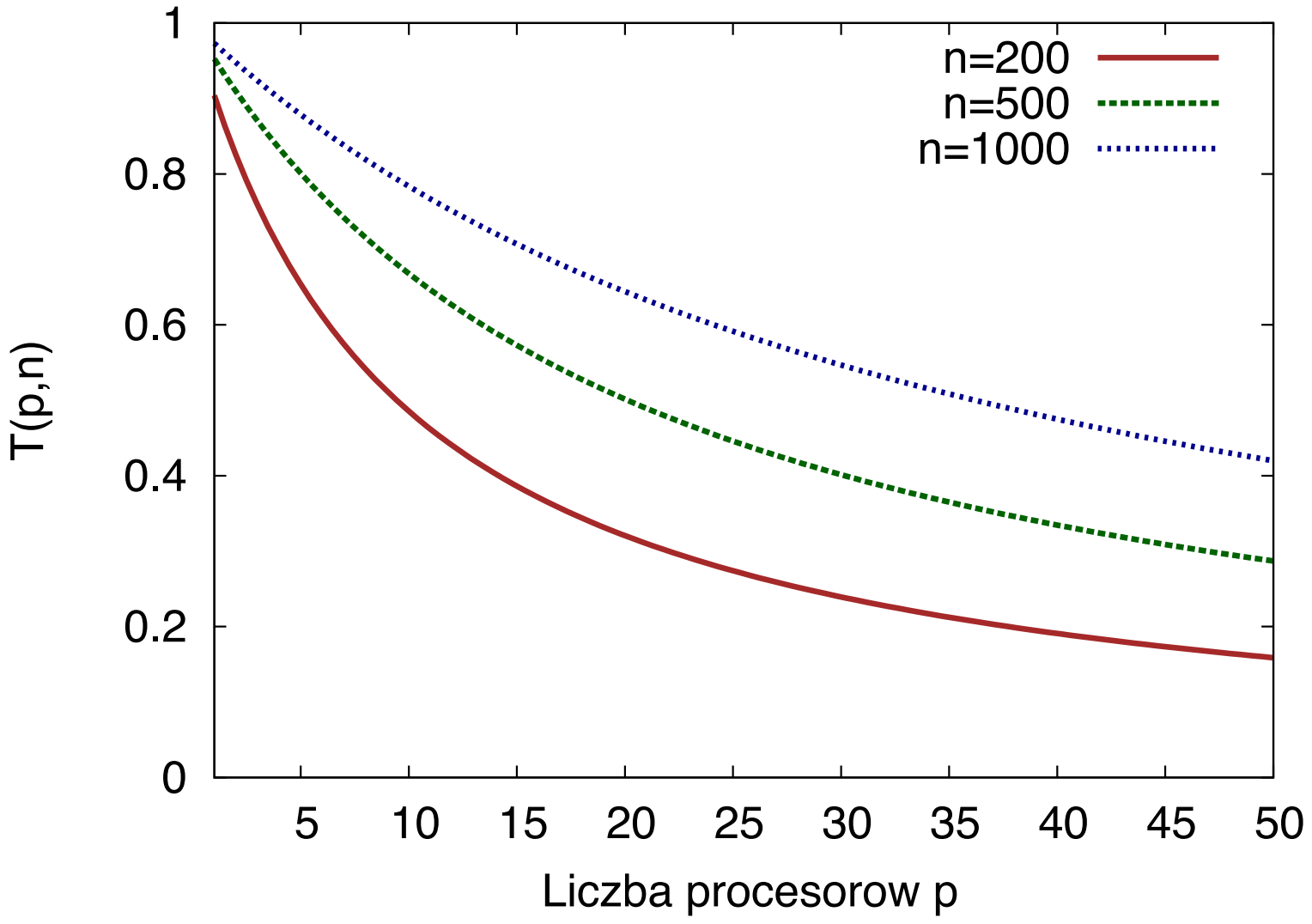
Złożoność


200
150
100
50
0
T(p,n)
5
10
15
20
25
30
35
40
45
50
Liczba procesorow p
n/p+log(n)
n/p
log(n)

Przyspieszenie



Efektywność



Algorytm wyznaczania sum prefiksowych

Dla danego zbioru $\{x_1, ..., x_n\}$ wyznaczyć wielkości:

$$\begin{aligned} s_1 &= x_1 \\ s_2 &= x_1 + x_2 \\ s_3 &= x_1 + x_2 + x_3 \\ &\dots \\ s_n &= x_1 + \dots + x_n \,. \end{aligned}$$

```
// Plik: prefixsum.alg
// Algorytm sekwencyjny (złożoność: T = O(n))
   s[1]=x[1];
   for (i=2; i<=n; i++)
      s[i]=s[i-1]+x[i];
-------------------------------------------------------
// Algorytm rownolegly:

 1 begin
 2    parfor P_i, 1 <= i <= n do\\
 3       s[i]=x[i];\\
 4    end parfor;
 5    for (j=0; j <= ceil(log n) - 1 do
 6       parfor P_i, 2^j <= i <= n do
 7          s[i] = s[i-2^j] + s[i];
 8       end parfor;
 9    end for;
10 end
```

Wyznaczanie elementu minimalnego w czasie O(1)

```
// Plik: min-o1.alg
// Dane:: tablica a[n]; n^2 procesorow; CRCW PRAM
// Zadanie: znalezienie elementu minimalnego w a (pierwszego najmniejszego)
-------------------------------------------------------------------------
 1 begin
 2   parfor P_i,j,  i=1, 1 <= j <=n do
 3      t[j] == 0;
 4   end parfor
 5   parfor P_i,j, 1 <= i,j <= n do
 6     if a[i] < a[j] then  // a[i] nie jest el. minimalnym
 7       t[j] = 1;
 8     end if;              // t[j]==0 znaczy, ze a[j] = min(a[j]), i=1,...,n
10     if t[i] == 0 and i<j then
11       t[j] = 1;
12     end if;              // tylko jeden element jest minimalny
13   end parfor;
14   parfor P_i,j, i=1, 1 <= j <= n do
15     if t[j] == 0 then
16       return a[j];      // zwrot elementu minimalnego
17     end if;
18   end parfor;
19 end;
```

Wyznaczanie elementu minimalnego w czasie O(1)

Algorytm działa w czasie $O(1)$!
Złożoność procesorowa $p(n) = n^2$, a więc cena, jest wysoka.
Koszt jest $O(n^2)$ i przewyższa koszt algorytmu sekwencyjnego, który wynosi $n - 1$.
Efektywność szybko maleje do zera wraz ze wzrostem n.

Sortowanie w czasie $O(\log n)$

Przeanalizujemy algorytm sortowania pracujący w czasie $O(1)$.
Zastosujemy tzw. sortowanie przez zliczanie. Użyjemy n^2 procesorów ułożonych w sieć $n \times n$. Problem sprowadza sie do wyznaczenia indeksu, pod który należy wstawić do $b[]$ każdy element tablicy $a[]$. Pomocnicza tablica obliczana przez wszystkie procesory jest dana przez:

$$w[i,j] = \begin{cases} 1 & \text{gdy } a[i] > a[j], \text{ lub gdy } a[i] = a[j] \text{ oraz } i > j \\ 0, & \text{w przeciwnym razie.} \end{cases} \tag{4}$$

Suma jedynek w każdym wierszu i tablicy w[] określa indeks elementu $a[i]$, pod który nalezy go wstawic do tablicy $b[]$ (suma jest mniejsza od indeksu o 1).
Jako przykład rozpatrzmy sortowanie tablicy $a[1..4] = \{5,-1,2,-1\}$. Relację opisaną przez (4) oznaczmy przez $\succ$. Wyniki przedstawia tabela.

$P_{i,j}$	1	2	3	4	w	1	2	3	4	$\sum$
1	$a[1] \succ a[1]$	$a[1] \succ a[2]$	$a[1] \succ a[3]$	$a[1] \succ a[4]$	1	0	1	1	1	3
2	$a[2] \succ a[1]$	$a[2] \succ a[2]$	$a[2] \succ a[3]$	$a[2] \succ a[4]$	1	0	0	0	0	0
3	$a[3] \succ a[1]$	$a31] \succ a[2]$	$a[3] \succ a[3]$	$a[3] \succ a[4]$	1	0	1	0	1	2
4	$a[14 \succ a[1]$	$a[4] \succ a[2]$	$a[4] \succ a[3]$	$a[4] \succ a[4]$	1	0	1	0	0	1

Sortowanie w czasie $O(\log n)$

```
// Plik: sort-Ologn.alg;    Algorytm sortowania w czasie O(lg n)
// Dane: Tablica a[n], n^2 = siec n x n procesorow, CREW PRAM
// Dane pomocnicze: w[1..n, 1..n];    Wynik: Tablica b[1..n] uporzadkowana;
 1 begin
 2   parfor P_ij, 1<= i,j <= n do
 3      if (a[i] == a[j]) or ((a[i] == a[j]) and (i>j)) then
 4         w[i,j] = 1;
 5      else
 6         w[i,j] = 0;
 7      end if;
 8   end parfor;
 9   k = n;
10   for (r=1; r<=ceil(log n) do  // zliczanie jedynek w tablicy w[]
11      parfor P_ij,  1 <= i <= n, 1 <= j <= floor(k/2) do
12         w[i,j] = w[i,j] +w[i, k+1-j];
13      end parfor;
14      k = ceil(k/2);
15   end for; // wartosci w[i,j] okreslaja pozycje elementow a[i] w tablicy b[]
16   parfor P_ij,  1 <= i <= n,  j=1 do
17      b[w[i,1]+1] = a[i];
18   end parfor;
19 end
```

Analiza złożonosci algorytmu sortowania przez n^2 procesorów

Wiersze 2–8 oraz 16–18 wykonują sie w jednym kroku (stały czas), a wiersze 9–15 w czasie $c\lceil \log n \rceil$ gdzie c jest stala. Mamy więc:
Złożoność czasowa:

$$T(p,n) = T(n) = O(\log n)$$

Przyspieszenie:

$$S(n) = \frac{n \log n}{\log n} = O(n)$$

Koszt:

$$C(n) = O(n^2 \log n)$$

Efektywność:

$$E(n) = \frac{n \log n}{n^2 \log n} = O(\frac{1}{n}).$$

Zalety i wady algorytmu sortowania

Zaletą algorytmu jest duże przyspieszenie. Wadą algorytmu jest mała efektywność.

DZIĘKUJĘ ZA UWAGĘ.