Obliczenia równoległe -podstawy teoretyczne

Wykład dla studentów
informatyki i bioinformatyki
2019/2020

por. Wprowadzenie do obliczeń równoległych, Zbigniew Czech 1.1-1.3

Procesy współbieżne

- **Program sekwencyjny** to ciąg instrukcji rozwiązujących problem. Realizowany może być przez jeden procesor.
- **Program współbieżny** składa się z pewnej liczby **zadań obliczeniowych** wchodzących w skład rozwiązania problemu obliczeniowego, które mogą być wykonywane równolegle (jednocześnie lub w tym samym czasie).
- **Zadanie obliczeniowe** jest realizowane przez proces sekwencyjny.
- **Proces sekwencyjny** to ciąg operacji obliczeniowych. Kolejność wykonywania operacji procesu jest ściśle określona. Początek rozpoczęcia kolejnej operacji procesu występuje nie wcześniej niż moment zakończenia operacji wcześniejszej.
- **Procesy współbieżne** to takie dwa lub więcej procesów sekwencyjnych, których operacje nakładają się w czasie (mogą nakładać się w czasie). Przyjmuje się, że procesy współbieżne są realizowane asynchronicznie – względny porządek operacji procesów może być dowolny.
- Gdy liczba **procesorów fizycznych** w systemie obliczeniowym jest **większa/równa** od liczby procesów współbieżnych to możliwe jest jednoczesne wykonywane wszystkich procesów współbieżnych . Na jednym procesorze fizycznym procesy współbieżne mogą być realizowane tylko:
  - metodą przeplotu po kolei operacje różnych procesów lub
  - sekwencyjnie po kolei operacje z jednego a następnie kolejnego procesu.

Komunikacja w systemach równoległych

Procesy współbieżne komunikują się ze sobą w celu przekazania danych i synchronizacji.

Komunikacja między procesami odbywa się:

- przy użyciu pamięci wspólnej (zmiennych wspólnych, współdzielonych) lub
- poprzez wymianę komunikatów.

Sposoby realizacji procesów współbieżnych

1. Procesy współbieżne realizowane są przez **jeden** procesor fizyczny metodą przeplotu (po kolei realizacja przez procesor różnych operacji różnych procesów).
2. Procesy współbieżne wykonywane przez **procesory** mające dostęp do **pamięci wspólnej**.
3. Procesy współbieżne realizowane przez **procesory rozproszone** (bez pamięci wspólnej) połączone przez kanały komunikacyjne dla potrzeb przesyłania komunikatów.

- **Program współbieżny** – zawiera procesy rozwiązujące dany problem obliczeniowy mogące być realizowane współbieżnie, synchronizujące się i komunikujące się , realizowane jednocześnie lub z przeplotem.
- **Program równoległy** – realizacja **programu współbieżnego** w ten sposób, że każdy proces współbieżny wykonywany jest przez odrębny procesor fizyczny.
- **Program rozproszony** występuje, gdy program równoległy jest realizowany przez procesory systemu rozproszonego.
- Wykonanie programu współbieżnego z przeplotem na jednym procesorze to **pseudorównoległy** sposób przetwarzania.

Zadania, procesy, wątki

- **Zadania** – części składowe programu równoległego wykonywane są pod nadzorem systemu operacyjnego jako **procesy sekwencyjne**. System operacyjny tworzy, nadzoruje i likwiduje proces. System operacyjny przydziela procesowi zasoby: czas procesora, obszar pamięci (na instrukcje, dane i stos).
- Proces jest realizowany jako jeden lub wiele wątków. **Wątek** to jednostka systemu operacyjnego realizująca strumień instrukcji. Wątki współdzielą przyznaną procesowi przestrzeń adresową określonego obszaru pamięci operacyjnej. Pamięć wspólna służy do komunikacji wątków.
- Proces i wątek - Proces ciężki i proces lekki (wątek, wspólne z innymi procesami zasoby – dane)

Poprawność programu sekwenyjnego

Program sekwencyjny jest poprawny gdy spełnione są warunki:

- **Częściowej poprawności** – dla określonych danych (spełniających założenia) programy kończące się dostarczą wymaganych wyników (stanu pamięci komputera);
- **Warunek stopu** - **pełna poprawność** występuje przy spełnieniu warunku stopu – gdy obliczenia zakończą się dla określonych danych (zakończą się instrukcje iteracyjne).

Poprawność programu współbieżnego

Warunkiem poprawności programu współbieżnego jest:

- poprawność procesów sekwencyjnych, które są jego realizacją oraz
- **własność bezpieczeństwa i żywotności**.

Własności bezpieczeństwa jest osiągalna poprzez **wzajemne wykluczanie** oraz **brak blokady** procesu składowego p.w. , a w przypadku grupy procesów – **brak zakleszczenia (**osiągane poprzez poprawną synchronizację procesów i poprawną organizację dostępów do zasobów współdzielonych**)**.

Wzajemne wykluczanie – w.w. procesów dotyczy dostępu do wspólnych plików, pamięci i realizacji na nich takich operacji jak zapis i uaktualnienie (celem w.w. jest zapewnienie **aktualności** używanych przez procesy danych).

Własność żywotności zapewnia możliwość **wystąpienia** warunku niezbędnego do wykonania każdego procesu składowego programu – nie wystąpi wtedy **zagłodzenie procesu** jako efekt jego nieskończonego oczekiwania na zasób wykorzystywany przez inne procesy.

Analiza efektywności przetwarzania współbieżnego

Źródła kosztów przetwarzania współbieżnego

- interakcje między procesami – komunikacja i synchronizacja;
- bezczynność – przyczyny:
  - brak zrównoważenia obciążenia (brak możliwości oceny czasu przetwarzania zadań),
  - obecność części sekwencyjnej przetwarzania,
  - konieczność synchronizacji;
- dodatkowe obliczenia –
  - algorytm równoległy dla swej efektywności wymaga innej konstrukcji od optymalnego algorytmu sekwencyjnego, np. powielenie obliczeń zamiast kosztownej komunikacji między procesami na różnych procesorach;

Miary efektywności systemów współbieżnych

- ***System współbieżny*** - połączenie algorytmu (programu) i architektury równoległej, w której jest on implementowany
- Czas przetwarzania (ang. runtime)
  - T_p – czas przetwarzania równoległego – od momentu rozpoczęcia przetwarzania równoległego do momentu zakończenia przetwarzania przez ostatnią jednostkę przetwarzającą
  - T_s – czas przetwarzania sekwencyjnego
- Koszt zrównoleglenia (ang. parallel overhead) – czas wspólnie spędzony przez jednostki współbieżne nad rozwiązywaniem problemu ponad czas niezbędny do rozwiązania tego samego problemu przez najlepszy algorytm sekwencyjny przy użyciu jednej jednostki przetwarzającej tego samego typu – $T_o = pT_p - T_s$

Przyspieszenie przetwarzania dzięki przetwarzaniu równoległemu

- Przyspieszenie – miara zysku wynikającego ze zrównoleglenia przetwarzania (realizowanego na *p* identycznych jednostkach przetwarzających) w stosunku do sekwencyjnego przetwarzania zrealizowanego przy użyciu najlepszego algorytmu sekwencyjnego, jednostki przetwarzające systemu równoległego są identyczne do wykorzystywanych w przetwarzaniu sekwencyjnym.

$$Sp = T^*_s / T_p$$

Maksymalne przyspieszenie przetwarzania

- Wartość przyspieszenia zazwyczaj nie przekracza liczby jednostek przetwarzających.
  - Nie spodziewamy się w ogólności wyższych wartości przyspieszenia gdyż:
  - *Załóżmy, że w wyniku podzielenia na **p** procesorów pracy - niezbędnej do rozwiązania problemu na jednym procesorze (realizowanej w czasie Ts), każdy z p procesorów będzie realizował przetwarzanie w czasie Tp równym co najmniej w czasie Ts/p - Tp>=Ts/p (najlepiej gdy podział równy i nie ma dodatkowych kosztów współpracy)*
  - *Jeżeli hipotetycznie miałoby być $Sp = T^*_s/T_p > p$ to (mnożymy przez Tp)*

    $$T^*_s > T_p * p$$
  - *Prawa strona nierówności to czas równy czasowi symulacji po kolei na jednym procesorze pracy T_p wykonanej jednoczesnie na **p** procesorach; czas ten byłby zgodnie z nierównością krótszy od najkrótszego czasu realizacji obliczeń sekwencyjnych. Nie jest to jednak możliwe, gdyż symulacja pracy wielu procesorów na 1 procesorze stanowi przetwarzanie sekwencyjne i nie może ono trwać krócej od **optymalnego** przetwarzania sekwencyjnego (z definicji optymalności). Założenie Sp>p prowadzi zatem do sprzeczności.*
  - *Po podstawieniu $T_p \geq T_s/p$ do $T^*_s > T_p * p$ otrzymujemy sprzeczność $T^*_s > T_s$*

Maksymalne przyspieszenie przetwarzania

W praktyce w niektórych systemach przyspieszenie większe od liczby procesorów może wystąpić.

Wartość Sp>p wynika z braku konieczności wykonania wszystkich obliczeń (tych samych co wykonane sekwencyjnie) w uruchomieniach współbieżnych przed zakończeniem przetwarzania.

Np. w przypadku algorytmów przeszukiwania (przeszukiwanie tabu, metoda podziału i ograniczeń…) celem programu jest znalezienie rozwiązania o określonych właściwościach. W użytym algorytmie może wystąpić kolejność przeszukiwania przestrzeni rozwiązań zależna od liczby procesorów. Czas przetwarzania programu nie jest warunkowany wtedy potrzebą wykonania określonej liczby obliczeń (zgodnie ze złożonością problemu – jak np. w algorytmie mnożenia macierzy) lecz zależy ostatecznie od momentu znalezienia rozwiązania spełniającego postawione wymagania i synchronizacji w systemie równoległym zakończenia przetwarzania.

Efektywność i koszt

Efektywność określa tę część czasu przetwarzania w jakiej procesory są efektywnie wykorzystane

$$E=Sp/p$$

W idealnym systemie równoległym $E=1$

Koszt (określany też jako praca) określa ilość czasu wykorzystywania procesorów systemu równoległego do rozwiązywania problemu:

$$C=T_p*p$$

Zakłada się tutaj korzystanie z procesorów od początku pracy systemu do uzyskania wyniku w ostatnim z pracujących procesorów.

Analizy
jakości algorytmów równoległych

por. Wprowadzenie do obliczeń równoległych, Zbigniew Czech 3.1-3.3

Złożoność algorytmu

Pesymistyczna złożoność czasowa algorytmu R (złożoność algorytmu) rozwiązującego problem o rozmiarze n przy użyciu p procesorów

$T(p,n) = Sup_{d \in Dn}\{ t(p,d) \}$

gdzie Sup – supremum jest wartością maksymalną spośród liczby kroków obliczeniowych dla różnych instancji o rozmiarze n. Kroki obliczeniowe te są określane od momentu rozpoczęcia pracy przez pierwszy z procesorów systemu równoległego do chwili zakończenia pracy przez ostatni z procesorów systemu równoległego. Jest to zatem liczba kroków (następujących po sobie) niezbędna dla najbardziej niesprzyjających danych wejściowych.

Przyspieszenie algorytmu

$$Sp(p,n) = T^*(1,n)/T(p,n)$$

Gdzie:

T(p,n) jest złożonością pesymistyczną algorytmu R,

T*(1,n) jest złożonością pesymistyczną najszybszego znanego algorytmu sekwencyjnego rozwiązującego ten sam problem co algorytm R.

Jest to **przyspieszenie bezwzględne** algorytmu R.

Przyspieszenie względne wystąpi przy zastąpieniu T*(1,n) przez T(1,n) (wtedy gdy nie użyto sekwencyjnego algorytmu najlepszego, lecz użyto np. algorytm R w „wersji sekwencyjnej” – uruchomiony na jednym procesorze.)

Kosztowo optymalny system współbieżny

System współbieżny jest *kosztowo optymalny* jeżeli koszt przetwarzania równoległego w funkcji rozmiaru problemu rośnie tak samo szybko (asymptotycznie - oznaczamy przez Θ) jak funkcja złożoności najlepszego algorytmu sekwencyjnego.

$rT^*(1,n) <= C(p,n) <= t\,T^*(1,n)$

dla stałych $r,t : 1<=r<=t$

Zależność oznaczamy $C(p,n)= \Theta(T^*(1,n))$

Wpływ granulacji obliczeń na efektywność

„Skalowanie w dół” systemu równoległego oznacza używanie mniejszej liczby procesorów (niż wymagana liczbą przetwarzanych danych i wynikająca z maksymalnego poziomu równoległości tego przetwarzania). Procesory zastosowane symulują pracę większej liczby jednostek przetwarzających.

W systemach optymalnych kosztowo skalowanie w dół nie powinno powodować spadku efektywności (powoduje wzrost ziarna przetwarzania – mniej kosztu komunikacji). Systemy nieoptymalne kosztowo mogą takimi pozostać, ale można też dzięki skalowaniu w dół uzyskać system kosztowo optymalny – jest to zależne od sposobu połączenia zadań w nowe zadania dla mniejszej liczby procesorów.

Z punktu widzenia zastosowań najbardziej interesujące są systemy optymalne kosztowo cechujące się najkrótszym czasem działania.

Skalowanie w dół systemu równoległego (metoda 1)

problem przykładowy: sumowanie 16 liczb na 4 węzłach

skalowanie: zamiast 8 procesorów – 4

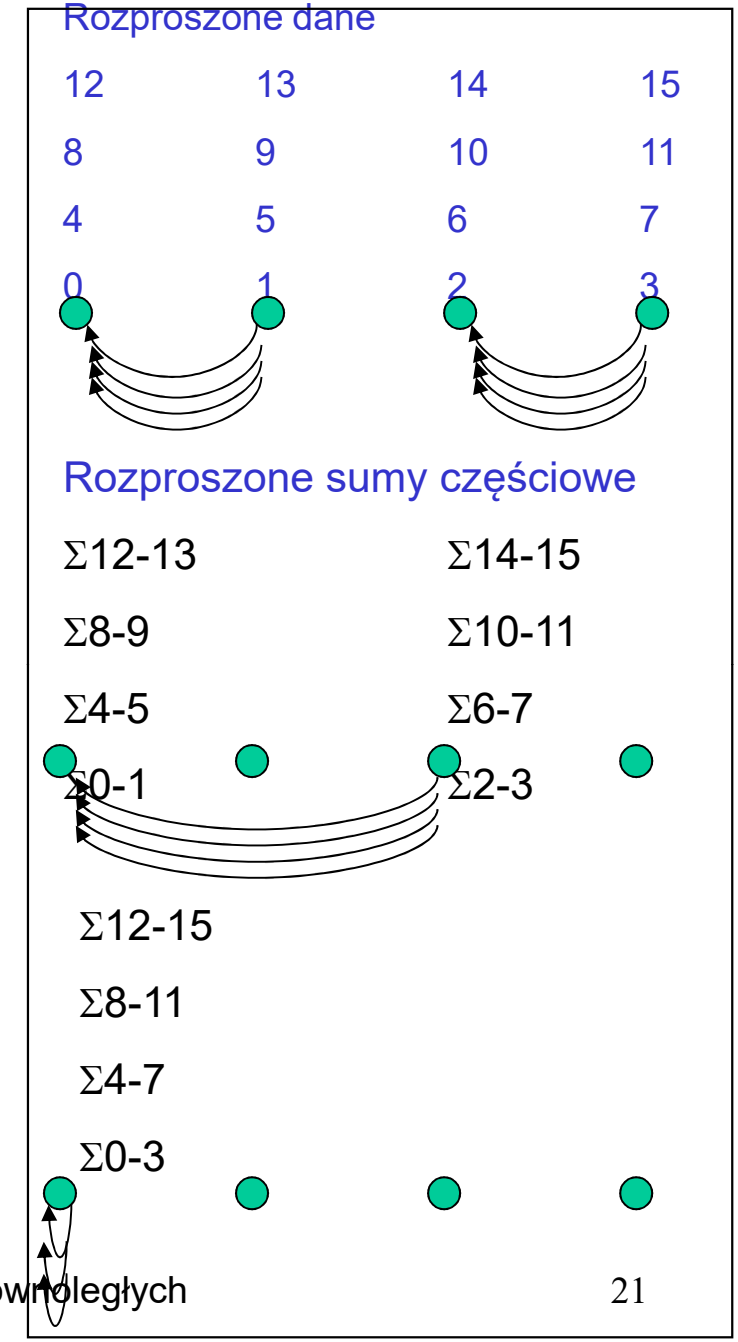
Metoda: prześlij i dodaj (nieefektywna)

$T_1 (p,n) = n/p \log p + n/p$

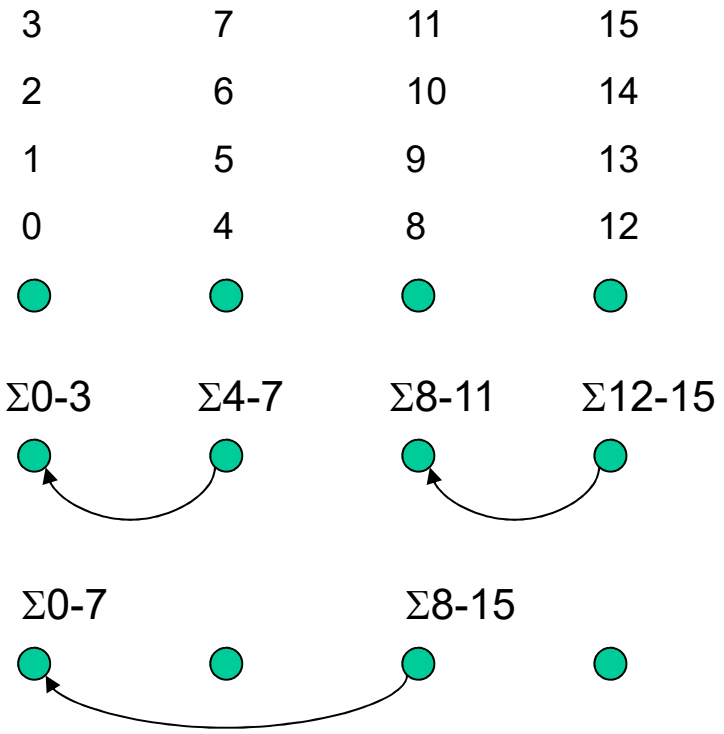
- *n/p przesłań i sumowań realizowane log p razy.*
- *Sumowanie w docelowym węźle to n/p*
- *OSTATECZNIE:*

$$C_1 (p,n) = T_1(p,n) * p = n \log p + n$$

$C_1 > \Theta(n)$ - system jest nieoptymalny kosztowo gdyż sekwencyjne sumowanie ma złożoność liniową.



Skalowanie w dół systemu równoległego (metoda 2)



Ten rodzaj skalowanie w dół systemu równoległego pozwala uzyskać system optymalny kosztowo.

Przetwarzanie realizowane w miarę możliwości lokalnie w procesorach.

1. $T_2(p,n) = n/p + \log p$; $C_2(p,n) = n + p \log p$
2. jeżeli n „asymptotycznie rośnie tak samo szybko jak” plog p

to $C_2(p,n) = \Theta(n)$, a system równoległy jest optymalny kosztowo

Skalowanie w dół jest konieczne, gdyż dla n=p system nie jest optymalny kosztowo.

Skalowalność systemów współbieżnych

System skalowalny umożliwia zachowanie stałej wielkości efektywności przy wzroście liczby procesorów dzięki wzrostowi rozmiaru przetwarzanego problemu.

Jak się system skaluje ? – jaki wzrost wielkości instancji jest potrzebny, aby przy wzroście liczby procesorów efektywność nie malała ?.

$$E = T(1,n) \ / \ (p * \ T(p,n))$$

$$T_o = p*T(p,n) - T(1,n) \ – \text{„koszt zrównoleglenia”}$$

$$\text{jeżeli} \quad p*T(p,n) = T(1,n) + T_o$$

$$\text{to} \quad E = 1/(1+T_o / T(1,n))$$

Koszt zrównoleglenia ≠ koszt przetwarzania równoległego (to inne pojęcia)

Stała efektywność f(p)

$$E = 1/(1+T_o/T(1,n))$$

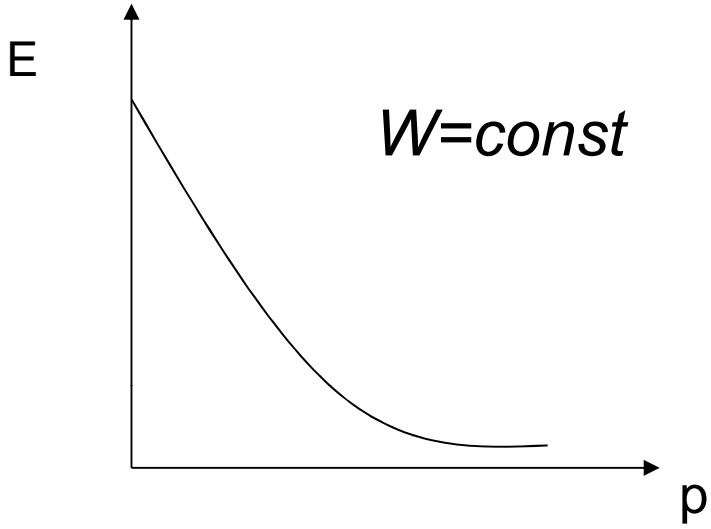
t_{serial} - *czas przetwarzania tej części, która się nie zrównolegla*

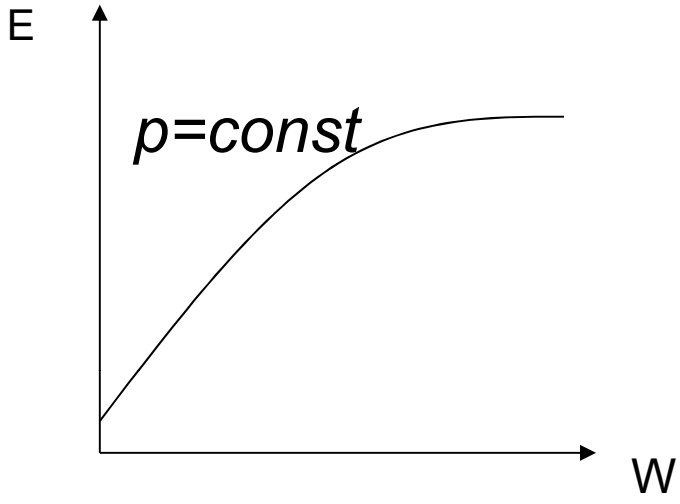
koszt zrównoleglenia $T_o > t_{serial}(p-1)$, gdyż np. część sekwencyjna realizowana jest tylko przez jeden procesor, p-1 pozostałych czeka;

Zatem koszt zrównoleglenia rośnie typowo z liczbą procesorów (np. wzrost kosztu komunikacji w większym systemie).

Aby zachować stałą efektywność przy wzroście liczby procesorów konieczny jest wzrost wielkości instancji pozwalający na wzrost poziomu równoległości równoważący wzrost kosztów interakcji, synchronizacji i komunikacji.

Typowe zależności efektywności f(p) f(W)





Funkcja stałej efektywności

T(1,n) = W – ilość operacji w algorytmie sekwencyjnym jest określona jako praca do wykonania

$$z\ E = 1/(1+T_o/\ T(1,n))$$

wynika $E= 1/(1+T_o(W,p)/W)$ *a stąd*

$$\boldsymbol{W= E/(1-E)\ T_o(W,p) \quad czyli \quad W= K\ T_o(W,p)}$$

- Wyznaczona z ostatniego równania funkcja W=f(p) jest **funkcją stałej efektywności** określającą:
- **jak łatwo** system równoległy zachowa stałą efektywność i osiągnie przyspieszenie odpowiednie do wzrostu liczby procesorów.
- jak powinna rosnąć ilość pracy do wykonania w systemie równoległym, aby zapewnić, że przy wzroście liczby procesorów zachowana zostanie wielkość efektywności.

Skalowalność systemów współbieżnych przykład

Dane z poprzedniego przykładu „system optymalny kosztowo dodawania liczb”

T(p,n)=n/p+2logp - złożoność przetwarzania równoległego - n/p sumowań lokalnych oraz log p sumowań i log p przesłań

T(1,n) = n - 1 ~ n (dla dużych n)

E = T (1,n) / T(p,n)/p = n/(n+2p logp) = 1/(1+2p logp/n)

Dla zachowania stałej efektywności (braku spadku efektywności)

2p *log p /n musi pozostać stałe (dokładniej - nie może rosnąć).

Dla p = 8 i n = 48 2p *log p /n = 1 i n = 2p log p

Dla p = 16 aby efektywność nie spadła potrzeba aby n >= 128 = 2*16 *log 16

Jeżeli liczba procesorów wzrośnie dwukrotnie z 8 do 16, to aby uzyskać nie gorszą efektywność rozmiar instancji problemu (sumowania w drzewie procesorów algorytmem optymalnym kosztowo) musi wzrosnąć nie mniej niż 8/3 razy (128/48)

Prawo Amdahla (1)

[Wprowadzenie do… R 3.2 3.3]

Założenia:

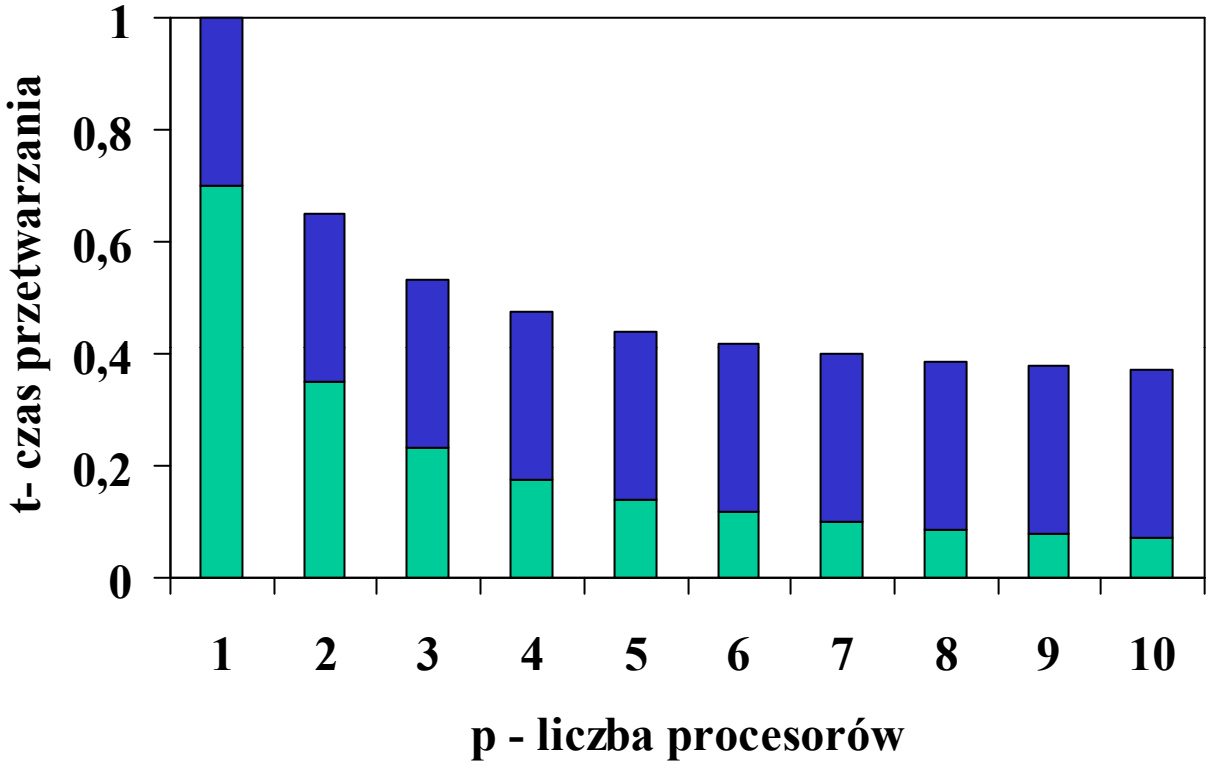
Obliczenia dla rozwiązania instancji problemu w sposób sekwencyjny T(1,n) mają dwa składniki:

- *s * T(1,n)* – możliwy do wykonania w sposób tylko sekwencyjny i
- *r *T(1,n)* – możliwy do wykonania w sposób równoległy. Daje się on zrównoleglić w dowolny sposób - czas przetwarzania maleje odwrotnie proporcjonalnie do użytej liczby procesorów.
- *s+r =1*

Rozpatrujemy zatem tutaj przetwarzanie instancji problemu o stałym rozmiarze.

Prawo Amdahla (2)

czas przetwarzania w funkcji liczby procesorów s=30%



Prawo Amdahla (3)

Sp(p,n)= (s + r) T(1,n) / (s T(1,n) + r T(1,n)/p)

Zatem:

$$Sp(p,n) = \frac{1}{s + \frac{r}{p}}$$

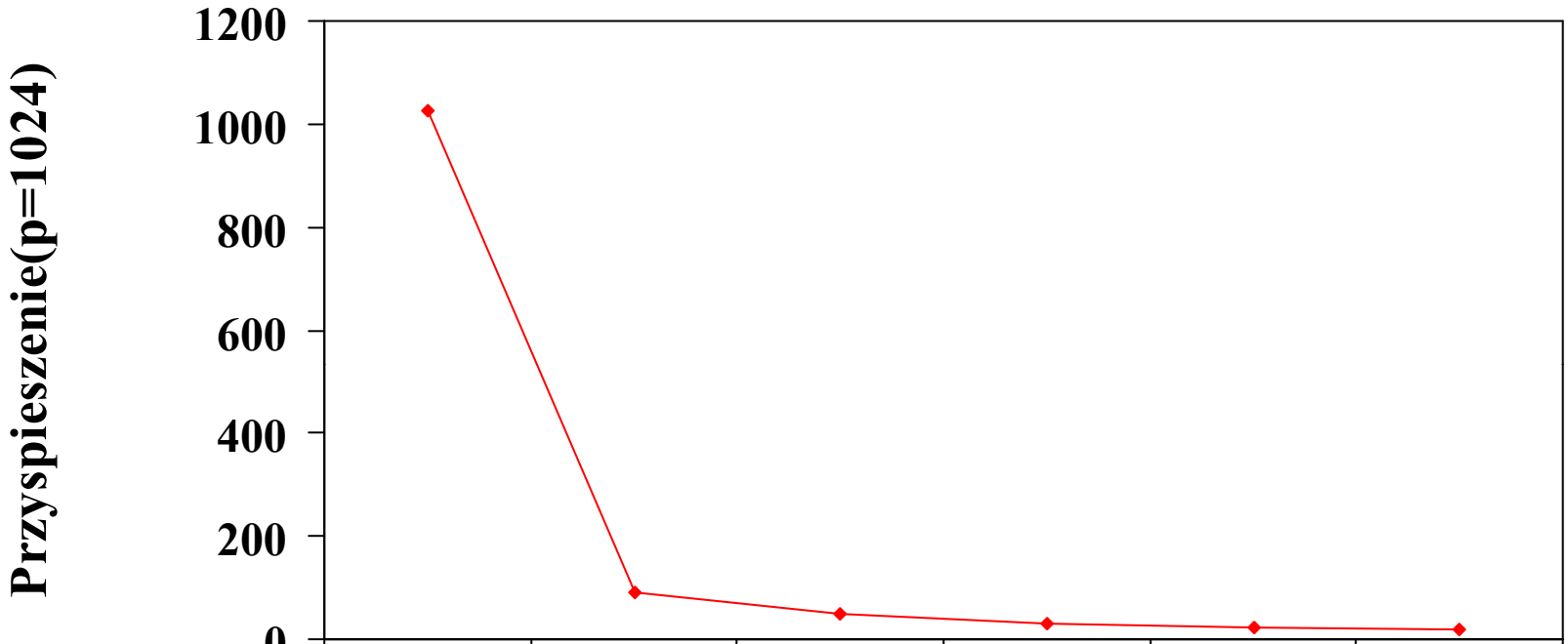
$$\lim_{p \to \infty} Sp(p) = 1/s$$

$$\lim_{r \to 1} Sp(p) = p$$

Prawo Amdahla (4)

przyspieszenie w funkcji wielkości części sekwencyjnej

s =<0%,5%>



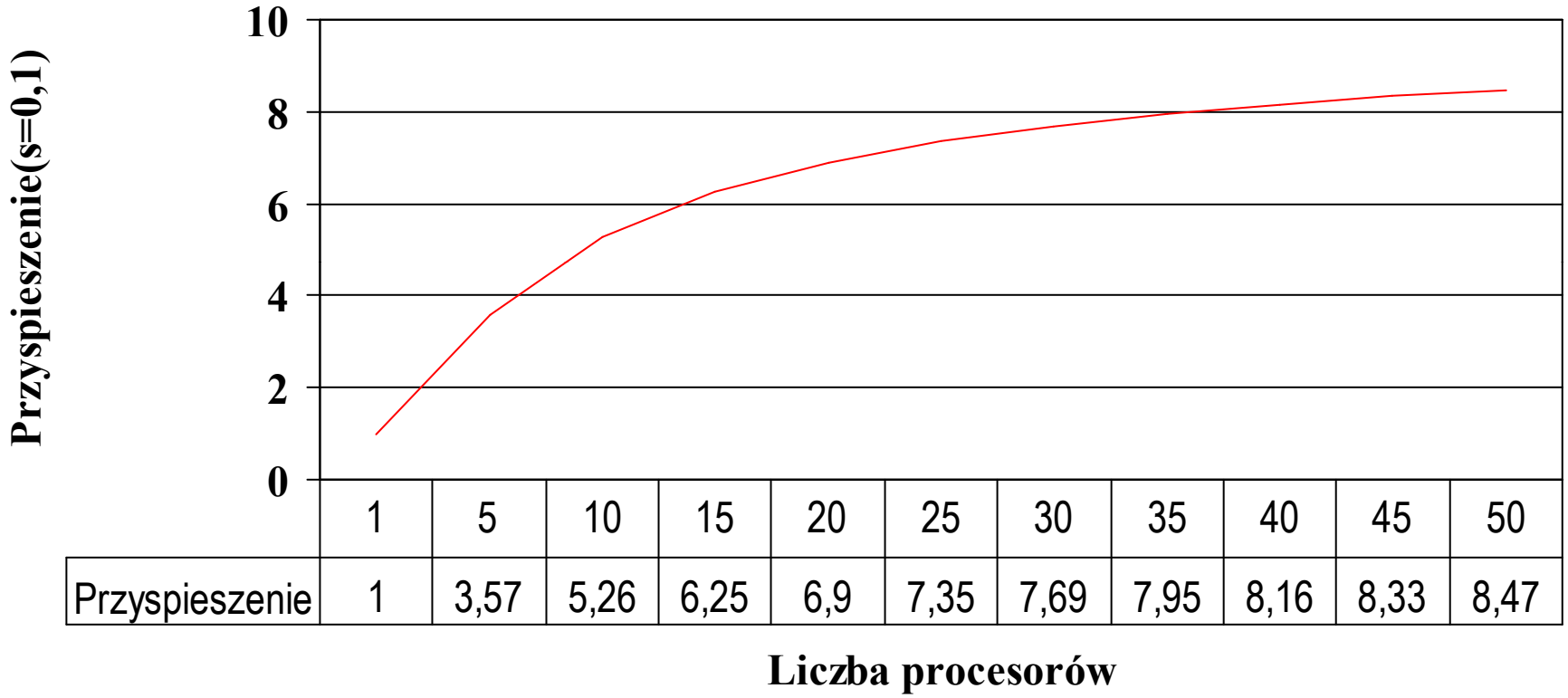
	0	0,01	0,02	0,03	0,04	0,05
PRZYSPIESZENIE	1024	91	48	32	24	20

s

Wybrane punkty wykresu połączone odcinkami prostymi dla większej czytelności

Prawo Amdahla (5)

przyspieszenie w funkcji liczby procesorów (s=10%)



	1	5	10	15	20	25	30	35	40	45	50
Przyspieszenie	1	3,57	5,26	6,25	6,9	7,35	7,69	7,95	8,16	8,33	8,47

Prawo Amdahla (6)

Wnioski:

- Ograniczenie stosowalności przetwarzania równoległego ze względu na ograniczenie wielkości możliwego do osiągnięcia przyspieszenia.
- Silny spadek przyspieszenia z niewielkim wzrostem wielkości części przetwarzania nie poddającej się zrównolegleniu.

Prawo Gustafsona (1)

Założenia:

Badamy przetwarzanie dla szeregu instancji problemu dla których czas wykonania programu w sposób sekwencyjny ma dwa składniki:

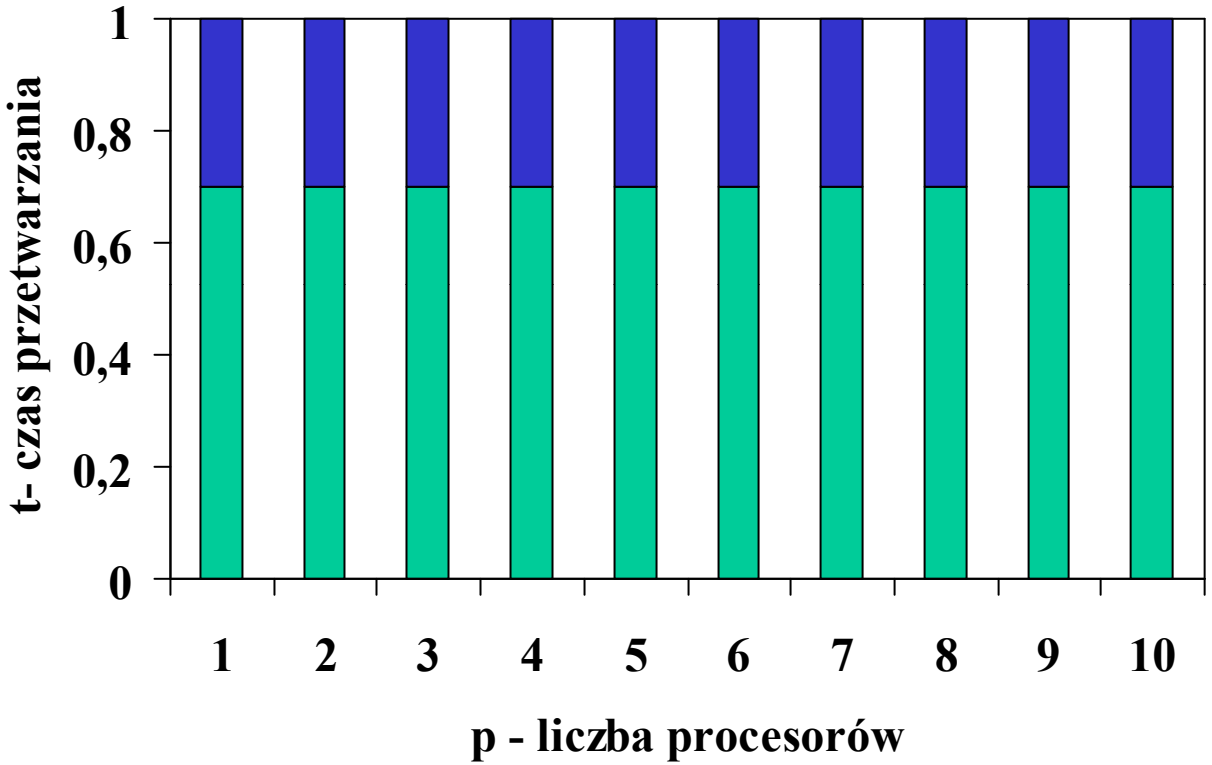
- ***s*** – część możliwa do wykonania jedynie w sekwencyjnie i
- ***p*r*** - (dla m = 1,2,3,...) – część możliwa do wykonania w sposób równoległy.
- *s*+*r* =1

Instancje o sekwencyjnym czasie przetwarzania T(1,n) = *s*+*r***p* są przetwarzane przy użyciu *p* procesorów (dla p= 1,2,3,...) w czasie s+r czyli T(p,n)= s+r

Rozpatrujemy zatem przetwarzanie szeregu instancji problemu o rosnącym rozmiarze i stałym czasie przetwarzania równoległego w funkcji użytych procesorów.

Prawo Gustafsona (2)

czas przetwarzania badanego szeregu instncji w funkcji liczby procesorów s=0,3



Prawo Gustafsona (3)

Wyznaczamy przyspieszenie dla wcześniej określonego szeregu instancji przetwarzanych na odpowiedniej liczbie procesorów

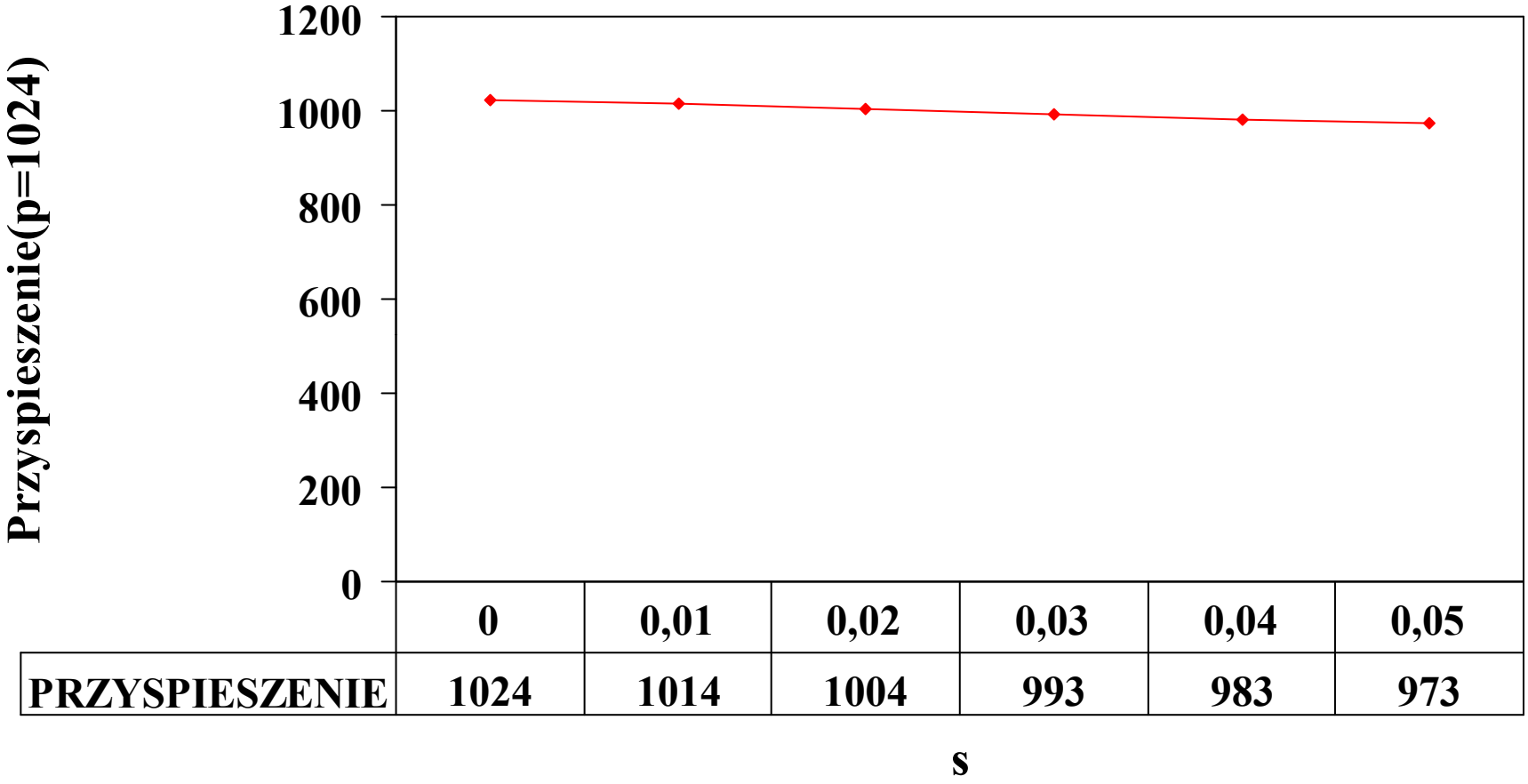
$$Sp(p) = \frac{s + pr}{s + r} = s + pr = p - s(p - 1)$$

$$\lim_{r \to 1} Sp(p) = p$$

Prawo Gustafsona (4)

przyspieszenie w funkcji wielkości części sekwencyjnej

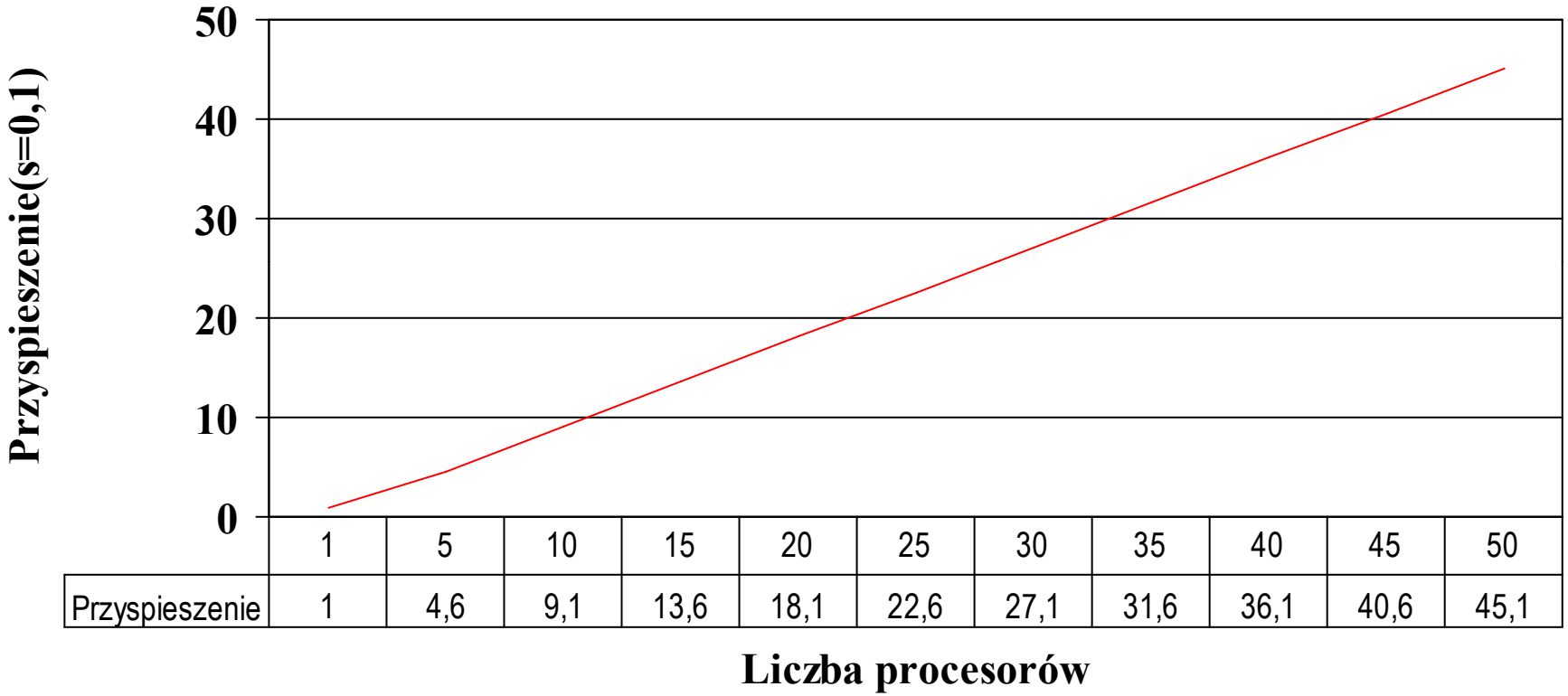
s =<0,0.05>



Wybrane punkty wykresu połączone odcinkami prostymi dla większej czytelności.

Prawo Gustafsona (5)

przyspieszenie w funkcji liczby procesorów s=0,1



	1	5	10	15	20	25	30	35	40	45	50
Przyspieszenie	1	4,6	9,1	13,6	18,1	22,6	27,1	31,6	36,1	40,6	45,1

Prawo Gustafsona (6)

Wnioski:

Przy wzroście złożoności przetwarzania ze wzrostem liczby procesorów możliwe jest efektywne przetwarzanie równoległe:

- Nie ma ograniczenia na wielkość możliwego do osiągnięcia przyspieszenia.
- Występuje liniowy wzrost przyspieszenia w funkcji liczby procesorów.
- Mniejszy wpływ wielkości **części** niepodlegającej zrównolegleniu na przyspieszenie. Założono niezależność wielkości części nie zrównoleglającej się od wielkości instancji problemu. Jest to wynik teoretyczny i w wielu przypadkach założenia nie są prawdziwe – np. wielkość części sekwencyjnej zależy od czasu trwania operacji wejścia-wyjścia, który to czas zazwyczaj rośnie z rozmiarem problemu (dla uzyskania wzrostu złożoności).

Model systemu równoległego - PRAM

[Wprowadzenie do… R 2.1 3.5]

PRAM – równoległy odpowiednik modelu RAM (model komputerowego przetwarzania sekwencyjnego), wspólna przestrzeń adresowa procesorów.

CECHY (4):

- jednoczesny dostęp każdego procesora do dowolnej komórki pamięci w jednostkowym czasie,
- Idealizacja maszyn równoległych z pamięcią współdzieloną,
- prosta analiza algorytmów,
- brak efektywnego algorytmu dla PRAM świadczy o braku efektywnego algorytmu dla dowolnej maszyny równoległej.

Maszyny RAM, PRAM

Maszyna RAM

- Jednostaka przetwarzająca i program
- Tasma wejściowa odczytu i tasma wyjściowa zapisu
- Rejestry (każdy mieści wartość integer dowolnego rozmiaru)
- Jednostkowy czas instrukcji

Maszyna PRAM – SM SIMD

- Wiele procesorów z rejestrami prywatnymi
- Wspólna pamięć (zamiast taśm)
- Możliwy dostęp równoległy do dowolnej komórki pamięci w jednostce czasu
- Wejście i wyjście algorytmu w określonych komórkach pamięci
- Synchroniczna realizacja instrukcji: przez procesory odczyt operandu, obliczenia i zapis
- Dostępy do pamięci EREW, CREW, ERCW, CRCW równoległe lub wyłączne.
- Współbieżny zapis wymaga arbitrażu w przypadku konfliktu dostępu; Możliwe strategie zapisu: jednolita, dowolna lub priorytetowa.

model PRAM

- Model pełnego równoległego dostępu do pamięci - nierealizowalny w praktyce ze względu na nierealność (wysoki koszt) zapewnienia *p* procesorom jednoczesnego dostępu do dowolnego słowa w pamięci (o *m* słowach), gdyż wymaga to *mp* przełączników sieci łączącej: równolegle pracujące porty pamięci (dla każdego słowa port) i procesory (dużo i drogo!!!).

Przykład: Algorytm dla CRCW PRAM z jednolitą strategią zapisu

Problem: Wyznaczenie maksimum elementów z tablicy TAB[m]

Jak to zrobić w 3 krokach ?

Przykład: Algorytm dla CRCW PRAM z jednolitą strategią zapisu

```
for i=1,m
    do parallel R[i]=TRUE; //początkowo każdy element i jest największy

 for i=1,m
     for j=1,m
                 do parallel  if (A[i]<A[j]) R[i]=FALSE;
 -- jeśli element jest mniejszy to otrzymuje FALSE
 -- jeśli procesor zapisuje to zapisuje FALSE – CW ok.
 -- R[i] dla wszystkich elementów największych (równych sobie) zachowuje TRUE
for i=1,m
     do parallel if (R[i]==TRUE) MAX= A[i];
```

Jeżeli zapisy dla różnych i np. i1 i i2 to istnieje tylko jedna wartość A[i1] = A[i2] (powielona na wielu pozycjach tablicy) dla której nie ma wartości większych.

Wykonanie programu przez m^2 procesorów zajmuje czas stały O(1). Jednoczesny zapis dotyczy jednakowych danych.

Sortowanie - algorytm PRAM (2)

Kroki przetwarzania;

- Porównanie każdej liczby z pozostałymi – jeden krok, n*n procesorów, zgodnie z: A[i]>A[j] lub (A[i]=A[j] i i>j) to IwiększeJ=1
- Scalenie n wyników porównania dla każdej liczby – efekt to informacja od ilu liczb badana liczba jest większa. Scalenie dla każdej liczby realizowane jest równolegle w $\log_2 n$ krokach, każda liczba wymaga n/2 procesorów, razem n*n/2 procesorów.
- Zapis liczb w pozycji zależnej od scalonego wyniku porównania - 1 krok, n procesorów.

Sortowanie - algorytm PRAM (2)

1. Ciąg wejściowy: 1,2,2,3
2. Porównanie dla 1: 0,0,0,0
3. Porównanie dla 2: 1,0,0,0
4. Porównanie dla 2: 1,1,0,0
5. Porównanie dla 3: 1,1,1,1
6. Pozycja dla 1: $\sum$=0
7. Pozycja dla 2: $\sum$=1
8. Pozycja dla 2: $\sum$=2
9. Pozycja dla 3: $\sum$=3
10. Wynik: 1,2,2,3

Inne zadania – algorytmy dla PRAM

- Sprawdzenie czy liczba jest pierwsza -

  Zapis do zmiennej faktu dzielenia bez reszty.
- Znalezienie największego dzielnika –

  zapis dzielników do tablicy i wyznaczenie maksimum.
- Znalezienie największego wspólnego dzielnika

  zapis znaczników o wspólnych dzielnikach do tablicy, scalenie tablic i wyznaczenie maksimum.