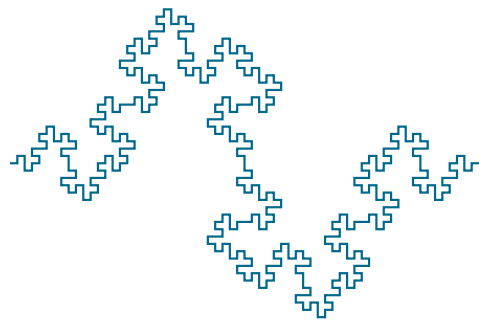
Programowanie współbieżne
Wstęp do obliczeń równoległych

Rafał Skinderowicz



Plan wykładu

- Modele obliczeń równoległych
- Miary oceny wydajności algorytmów równoległych
- Prawo Amdahla
- Prawo Gustavsona

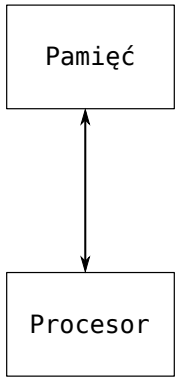
Modele obliczeń równoległych

- Projektując algorytm rozwiązujący zadany problem należy uwzględnić model obliczeń
- Model obliczeń określa w jaki sposób mogą być gromadzone, przechowywane i przetwarzane dane i wyniki obliczeń oraz jakie operacje mogą być wykonywane na danych
- Modele obliczeń mogą różnić się siłą
- Nieformalnie siła modelu określa jakiego rodzaju problemy można rozwiązać za pomocą algorytmu realizowanego w danym modelu obliczeń

Modele obliczeń równoległych

- Siłę modelu można wyrazić za pomocą rozpoznawanej klasy języków formalnych
- Przykładowe modele obliczeń wg rosnącej siły:
  - Deterministyczny automat skończony – rozpoznaje języki regularne
  - Automat ze stosem – rozpoznaje języki bezkontekstowe
  - Deterministyczna Maszyna Turinga oraz model RAM – rozpoznają języki rekursywne
- Najpopularniejszy model obliczeń, to RAM, czyli maszyna o dostępie swobodnym (ang. Random Access Machine)

Model RAM



Model RAM składa się z:

- Pamięci złożonej z nieskończonej liczby komórek o unikalnych *adresach*
- W komórce pamięci można zapisać liczbę całkowitą
- Czas dostępu (zapis, odczyt) do komórki *niezależnie* od jej adresu jest stały
- Procesor realizuje program złożony z ciągu instrukcji (należących do pewnego skończonego zbioru)
- Program zapisany jest w jednostce sterującej modelu, a nie w pamięci

Model RAM

- Model obliczeń jest wygodną abstrakcją rzeczywistej maszyny obliczeniowej
- Ułatwia projektowanie i *analizę* algorytmów
- Podstawowymi kryteriami oceny algorytmu sekwencyjnego dla modelu RAM są:
  - Złożoność czasowa – funkcja określająca czas działania algorytmu w zależności od rozmiaru danych wejściowych
  - Złożoność pamięciowa – funkcja określająca liczbę komórek pamięci niezbędną do poprawnego wykonania programu w zależności od rozmiaru danych wejściowych
- Rozmiar danych wejściowych, to liczba naturalna określająca wielkość zbioru danych wejściowych, np. w problemie sortowania ciągu liczb równa jest liczbie elementów w ciągu

Złożoność czasowa – przypomnienie

Najczęściej wyróżnia się złożości czasowe:

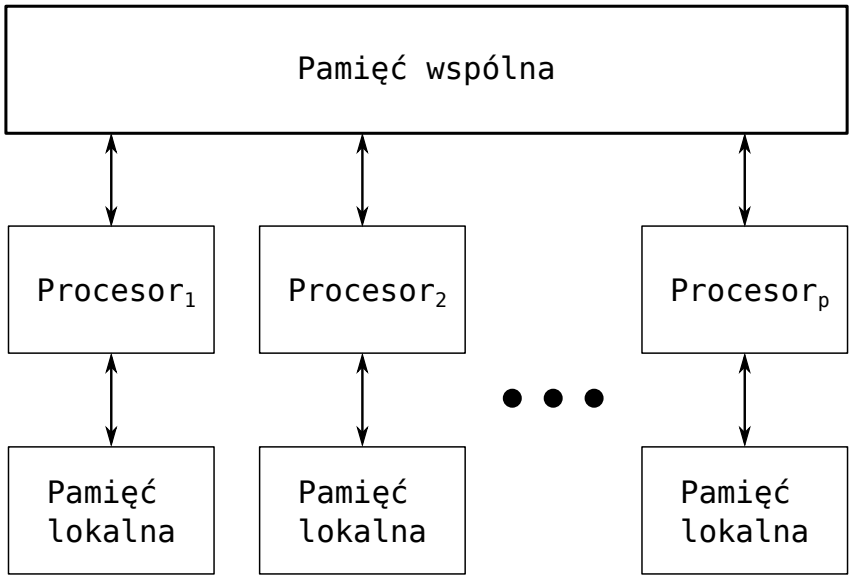
- pesymistyczną – określa najdłuższy możliwy czas działania algorytmu dla danych wejściowych o danym rozmiarze n
  $T_{\text{pes}}(n) = \sup_{d \in D_n}\{t(d)\}$
- oczekiwaną (średnią) – określa oczekiwany czas działania algrytmu dla danych wejściowych o rozmiarze n
  $T_{\text{śr}}(n) = \sum_{d \in D_n} p(d)t(d)$,
  gdzie D_n to zbiór zestawów danych wejściowych,
  $p(d)$ – prawd. wystąpienia zestawu

Złożoność czasową wyraża się przez liczbę wykonywanych w algorytmie dominujących operacji elementarnych ($t(d)$)

Modele obliczeń równoległych

- Dla obliczeń równoległych definiuje się odrębne modele obliczeń
- Do najpopularniejszych należą:
  - model równoległy z pamięcią wspólną (ang. parallel random access machine, PRAM)
  - model sieciowy – dla obliczeń z przesyłaniem wiadomości

Model PRAM



- Model wzorowany na modelu RAM
- Procesory współdzielą pamięć, ale mają również pamięć lokalną
- Procesory pracują synchronicznie – centralny zegar
- Programy procesorów mogą się różnić
- Pamięć wspólna umożliwia szybkie komunikowanie się procesorów ze sobą – komórka zapisana w kroku i przez dany procesor może zostać odczytana w kroku $i + 1$ przez inny

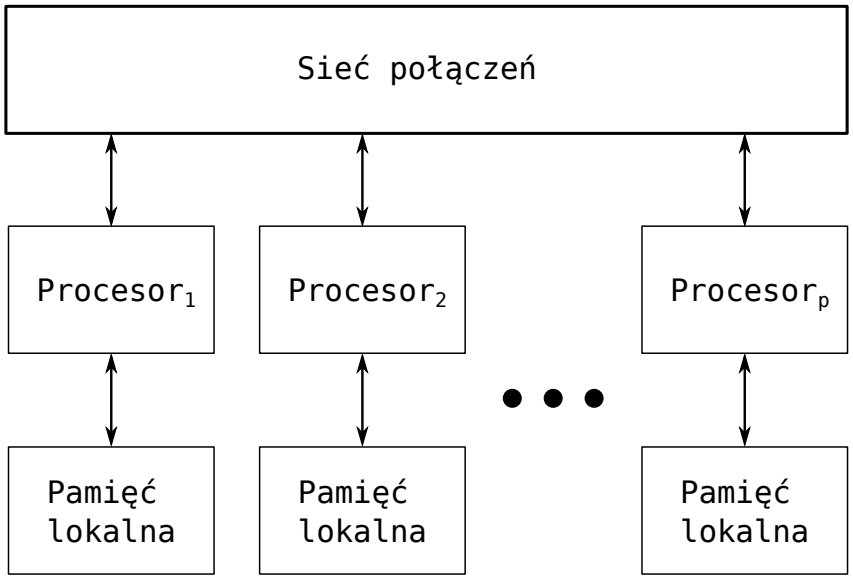
Warianty modelu PRAM

- W modelu PRAM mogą wystąpić konflikty w dostępie do pamięci wspólnej, np. jeden procesor zapisuje, a drugi odczytuje w tym samym kroku tę samą komórkę
- W związku z tym wyróżnia się następujące *warianty* modelu PRAM:
  - EREW (ang. exclusive read, exclusive write) – niedopuszczalny jest jednoczesny dostęp (ani zapis, ani odczyt) do tej samej komórki
  - CREW (ang. concurrent read, exclusive write) – dopuszczalny jest jednoczesny odczyt przez wiele proc., ale nie zapis
  - CRCW (ang. concurrent read, concurrent write) – dopuszcza się zarówno jednoczesny odczyt, jak i zapis przez wiele procesorów
    - Wyróżnia się warianty CRCW ze względu na sposób rozwiązania konfliktu w przypadku jednoczesnego zapisu

Wady i zalety modelu PRAM

- Podstawową zaletą modelu PRAM jest jego prostota, która ułatwia analizę algorytmów projektowanych dla tego modelu, tj. dowodzenie poprawności, analiza złożoności
- Podstawowy problem, to brak możliwości praktycznej realizacji
  - Nie da się utrzymać *stałego* czasu dostępu do pamięci, gdy liczba procesorów wzrasta
  - W praktyce udaje się implementować dla maks. kilkudziesięcu procesorów

Model sieciowy



- Brak współdzielonej pamięci – w zamian procesory (węzły) połączone łączami
- Łącza tworzą sieć połączeń
- Wymiana danych między procesorami wymaga *przesyłania komunikatów* (ang. message passing)
- Procesory mogą pracować synchronicznie lub asynchronicznie

Model sieciowy – topologie

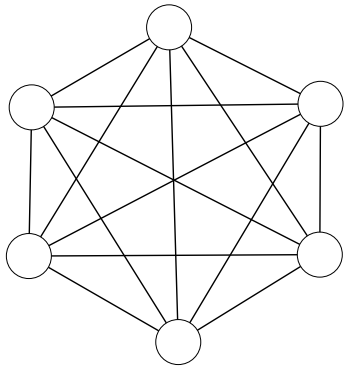
Sposób połączenia procesorów ze sobą definiuje topologię sieci połączeń. Do jej opisu wystarczy graf:

- wierzchołki to procesory
- krawędzie to dwukierunkowe łącza między procesorami

Nie każda topologia nadaje się do każdego rodzaju obliczeń równoległych.

Różne topologie sieci połączeń można opisać parametrami:

- średnica sieci
- maksymalny stopień wierzchołka
- szerokość połowienia sieci
- spójność krawędziowa
- koszt sieci



Rysunek : Sieć pełna

Parametry sieci

- Średnica sieci to maksymalna liczba łączy między dowolną parą wierzchołków – im mniejsza średnica tym lepiej
- Maksymalny stopień wierzchołka – maksymalna liczba łączy do danego procesora
  - wpływa na złożoność procedur komunikacyjnych – im mniejszy stopień tym lepiej
  - przy wzroście liczby proc. nie powinien rosnąć szybko – najlepiej wcale
- Szerokość połowienia sieci – minimalna liczba krawędzi, które muszą zostać usunięte, aby podzielić ją na dwie równe podsieci
  - od szerokości połowienia zależy przepustowość sieci, a od niej wydajność
  - większa szerokość połowienia, to mniejsza liczba konfliktów w komunikacji

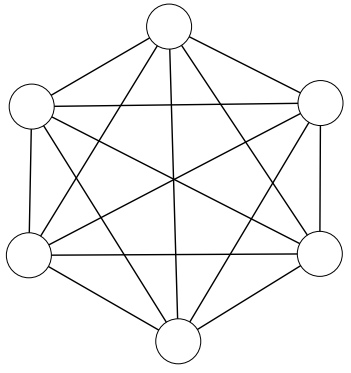
Parametry sieci, c.d.

- Spójność krawędziowa – minimalna liczba krawędzi, które muszą ulec awarii, aby sieć stała się *niespójna*
  - im większa spójność, tym większa *odporność na uszkodzenia*
  - duża spójność to mnogość dróg między procesorami
- Koszt sieci – koszt wykonania, zarządzania i utrzymania sieci połączeń między procesorami
  - w najprostszym przypadku mierzony liczbą krawędzi (łącz)
  - w rzeczywistości trzeba uwzględnić przełączniki, karty sieciowe itd.

Parametry sieci połączeń – przykład

Najlepsze własności komunikacyjne ma *sieć pełna*

- średnica sieci: 1
- maksymalny stopień wierzchołka: $p - 1$
- szerokość połowienia sieci: $p^2/4$
- spójność krawędziowa: $p - 1$
- koszt sieci: $p(p - 1)$



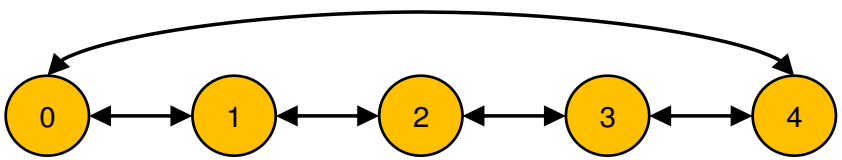
Rysunek : Sieć pełna

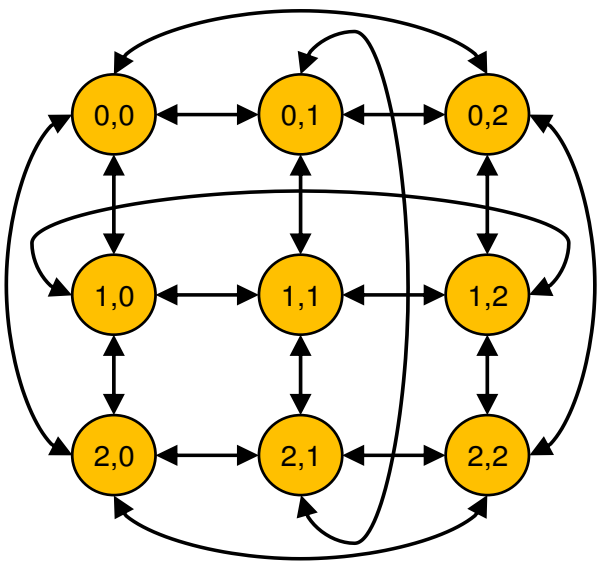
Podstawowe wady:

- duży koszt dla 100 tyś. proc. potrzeba blisko 10 mld (!) łączy
- fatalna skalowalność

Siatki i torusy

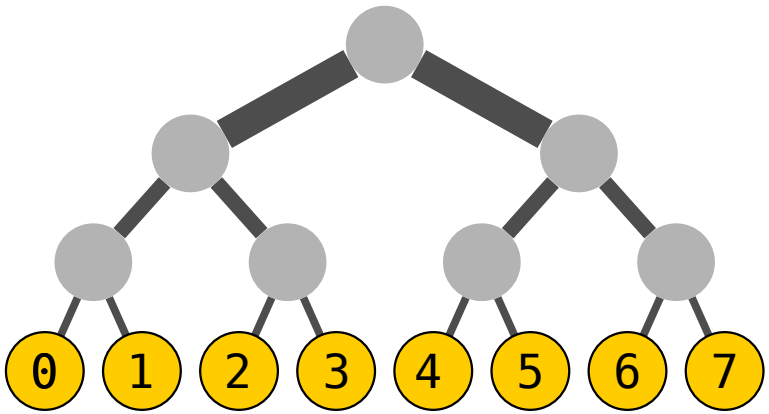
- Do najpopularniejszych topologii sieci połączeń należą siatki
- Kompromis – relatywnie niski koszt, ale zwiększona średnica i niska spójność krawędziowa
- W siatce d–wymiarowej każdy procesor oprócz „brzegowych” jest połączony z $2d$ procesorami (stopień wierzchołka)
- Po dodaniu połączeń między brzegowymi procesorami otrzymujemy d–wymiarowy torus





Fat-tree

- Procesory stanowią liście drzewa, węzły wewnętrzne to przełączniki
- „Szerokość” łączy rośnie im bliżej korzenia – stała szerokość połowienia na każdym poziomie drzewa – ang. constant bi-sectional bandwidth (CBB)
- Umożliwia nieblokujące połączenia między dowolną parą węzłów – duża wydajność



Parametry dla wybranych topologii sieci

Rodzaj sieci	**Średnica**	**Maks. stopień**	**Szer. poł.**	**Spójność kraw.**	**Koszt**
Pełna	1	$p-1$	$p^2/4$	$p-1$	$p(p-1)/2$
Siatka 1-d	$p-1$	2	1	1	$p-1$
Pierścień	$\lfloor p/2 \rfloor$	2	2	2	p
Siatka 2-d	$2(\sqrt{p}-1)$	4	$\sqrt{p}$	2	$2(p-\sqrt{p})$
Torus 2-d	$2\lfloor \sqrt{p}/2 \rfloor$	4	$2\sqrt{p}$	4	2p
Kostka	$\log p$	$\log p$	$p/2$	$\log p$	$(p \log p)/2$
Fat tree	$2 \log_2 p$	p	$p/2$	1	$p \log_2 p$

Podsumowanie

- Projektując algorytm równoległy należy uwzględnić *architekturę* komputera, na którym będzie wykonywany
- Idealnie, gdy zadanie daje się podzielić na *niezależne* porcje obliczeń o podobnym i odpowiednio *dużym* rozmiarze
- Każdy okres czasu poświęcony na oczekiwanie na transfer danych obniża efektywność wykorzystania procesorów, nierzadko drastycznie
- Nie każde zadanie nadaje się dla każdej architektury, szczególnie w modelu sieciowym – ograniczona przenośność

Złożoność czasowa alg. równoległych

- Dodatkowa zmienna – liczba procesorów p
- Wzór:

$$T_{\text{pes}}(p, n) = \sup_{d \in D_n} \{t(p, d)\}$$

gdzie:

- D_n to zbiór zestawów danych wejściowych,
- $t(p, d)$ – liczba operacji dominujących wykonywanych dla zestawu danych d od momentu rozpoczęcia obliczeń przez pierwszy procesor, aż do momentu zakończenia obliczeń przez wszystkie procesory

Przyspieszenie

Przyspieszenie algorytmu równoległego definiuje się następująco:

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

gdzie $T^*(1,n)$ oznacza złożoność czasową najlepszego algorytmu *sekwencyjnego* dla rozwiązywanego problemu

- Przyspieszenie określa *korzyść* z wykorzystania obliczeń równoległych
- Maksymalna wartość przyspieszenia to p – przyspieszenie *liniowe*

Przyspieszenie c.d.

- Zazwyczaj $S(p, n)$ jest mniejsze od p ze względu na nierównomierne obciążenie procesorów obliczeniami
- W rzeczywistości, w rzadkich przypadkach, można uzyskać przyspieszenie superliniowe, na skutek zwiększonego sumarycznego rozmiaru pamięci podręcznej

Złożoność c.d.

W złożoności czasowej $T(1, n)$ można wyróżnić obliczenia, które:

- muszą być wykonane sekwencyjnie – $T^s(1, n)$
- mogą być wykonane równolegle – $T^r(1, n)$
- czyli $T(1, n) = T^s(1, n) + T^r(1, n)$

Konieczność sekwencyjnego wykonania obliczeń wynika najczęściej z zależności danych

```
x:= a + b;
y:= c * x ;
```

```
x:= a + b;
y:= c * d;
```

Przyspieszenie c.d.

Zakładając, że obliczenia da się *równo* rozdzielić między procesorami otrzymujemy:

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

- $T^o(1,n)$ jest dodatkową złożonością wynikającą z *organizacji* obliczeń równoległych, na którą składa się m.in. komunikacja między procesorami
- $T^o(1,n)$ rośnie wraz z liczbą procesorów p
- $T^r(1,n)$ zazwyczaj rośnie wraz z n szybciej, niż $T^o(1,n)$

Koszt algorytmu równoległego

Oprócz przyspieszenia czasem podaje się powiązaną miarę kosztu algorytmu, której definicja to:

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

- Koszt jest sumą kroków obliczeniowych wykonywanych przez wszystkie procesory podczas rozwiązywania problemu
- Idealnie, gdy procesory w algorytmie równoległym wykonują łącznie dokładnie tyle samo operacji, co jeden procesor w algorytmie sekwencyjnym: $pT(p,n) = T^*(1,n)$
- W praktyce jest to trudne, ponieważ każda dodatkowa operacja poświęcona np. na komunikację między procesorami jest już nadmiarowa w stosunku do $T^*(1,n)$

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

Efektywność

Efektywność wykorzystania procesorów (ang. efficiency, processor utilization) definiowana jest jako:

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

- Maksymalna efektywność wynosi 1, co oznacza, ze procesory wykorzystywane są w 100% – czyli brak okresów bezczynności oraz zerowe koszty komunikacji

Skalowalność

- Nieformalnie skalowalność odpowiada na pytanie, czy *opłacalne* jest zwiększenie liczby procesorów (np. podwojenie), by skrócić czas działania algorytmu
- *Opłacalność* można mierzyć efektywnością obliczeń, a więc zależy nam na zachowaniu stałej efektywności
- Efektywność maleje wraz ze wzrostem liczby procesorów, ponieważ rosną m.in. koszty komunikacji

Skalowalność c.d.

- Przykładowo:
  - załóżmy, że dla $p = 10$ procesorów efektywność $E(10, n) = 80\%$, czyli przyspieszenie $S(10, n) = pE(10, n) = 8$,
  - po podwojeniu liczby procesorów efektywność spadła do 60%, czyli $E(20, n) = 60\%$, stąd przyspieszenie $S(20, n) = 12$
  - czy opłaca się za skrócenie o połowę czasu obliczeń zapłacić dwukrotnie więcej?

Skalowalność c.d.

Załóżmy, że dana jest równoległa wersja algorytmu sekwencyjnego o efektywności

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

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

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

Skalowalność c.d.

W celu zachowania stałej efektywność przy *rosnącej* liczbie procesorów należy zachować stałą wartość ułamka:

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

- Gdy „dokładamy” procesory, to nieuchronnie rośnie $C^o(p,n)$
- Pozostaje nam zwiększanie $T(1,n)$, które zależy od n
- Czyli musimy zwiększać rozmiar danych wejściowych
- Innymi słowy – *większe komputery dla większych problemów*

Jeżeli udaje się zachować efektywność przy rosnącej liczbie procesorów oraz rozmiarze problemu, to algorytm (system) jest skalowalny

Przyspieszenie – uwagi

- Jak wspomniano, wartość przyspieszenia zależy przede wszystkim od tego jaką część obliczeń w algorytmie można wykonać równolegle, a jaka musi zostać wykonana sekwencyjnie.

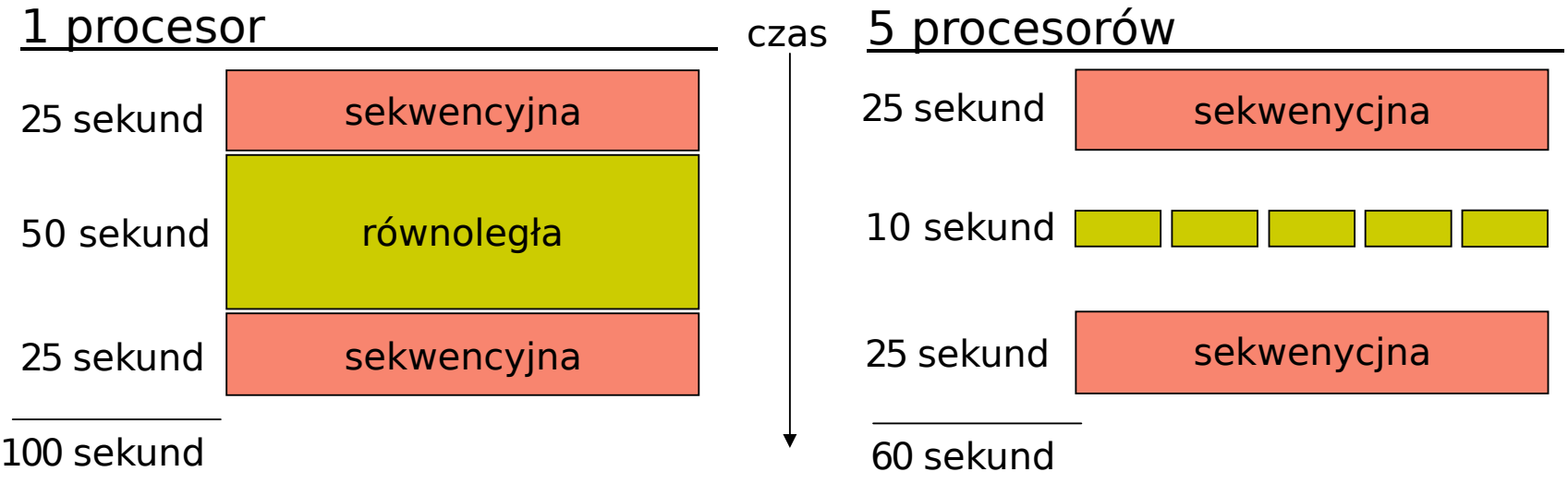
część sekwencyjna (zależność danych)

```
a = b + c;
d = a + 1;
e = d + a;
```

część równoległa (brak zależności danych)

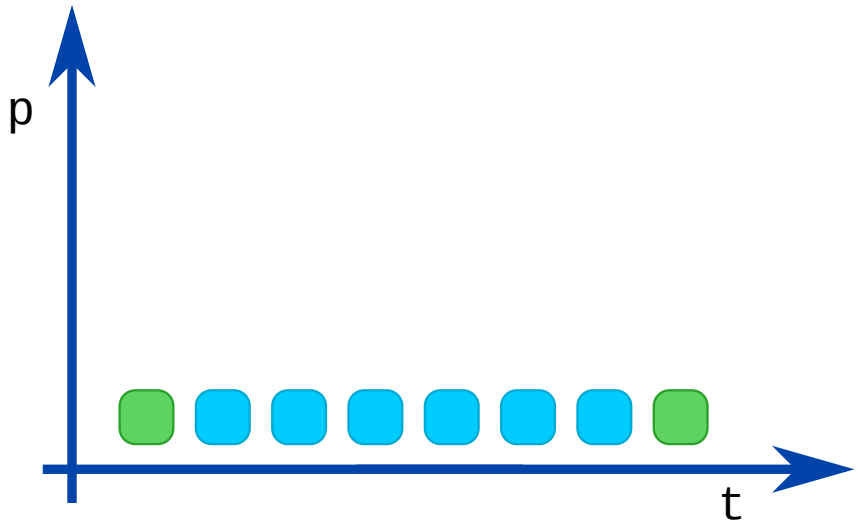
```
for(i=0; i < e; i++)
    M[i] = 1;
```

Przyspieszenie – przykład

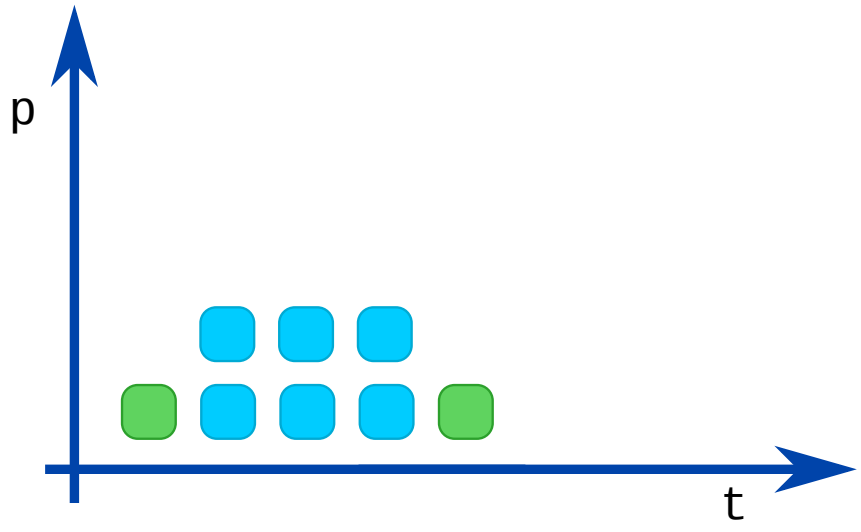


- Przyspieszenie = 100 sek. / 60 sek. = 1.67 razy
- Efektywność = 1.67 / 5 = 33.4%

Przyspieszenie – przykład

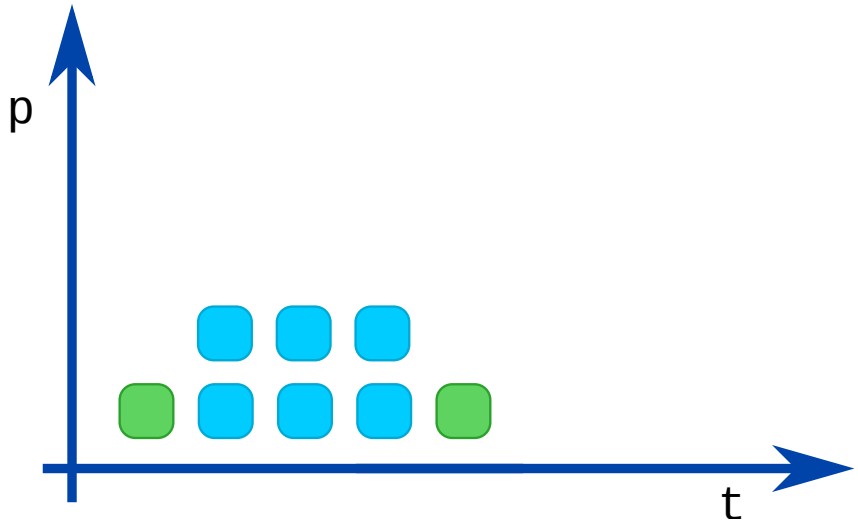


- Rozmiar danych $n = 8$
- Czas wykonania $T(1, n) = 8$

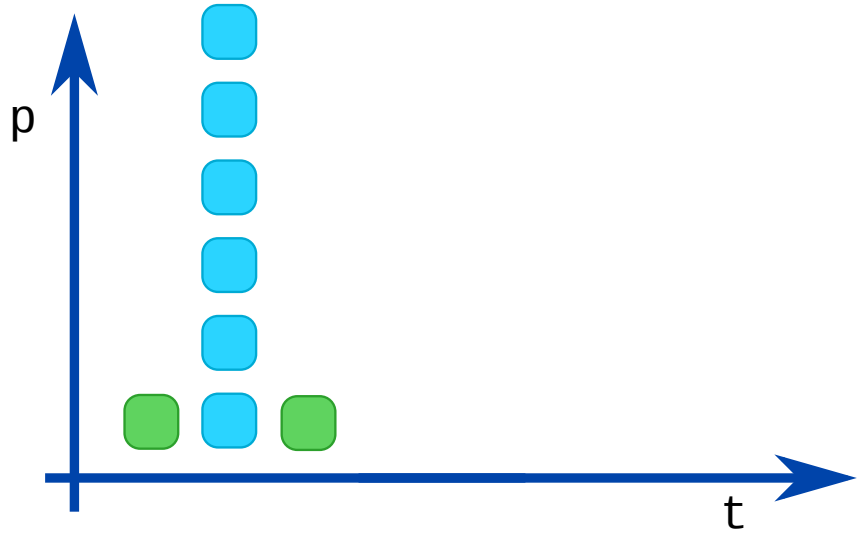


- Czas wykonania $T(2, n) = 5$
- Przyspieszenie $S(2, n) = \frac{T(1,n)}{T(2,n)} = \frac{8}{5}$
- Efektywność $E(2, n) = \frac{S(2,n)}{2} = \frac{8/5}{2} = \frac{8}{10}$
- Koszt $C(p, n) = p \cdot T(p, n) = 2 \cdot T(2, n) = 10$

Przyspieszenie – przykład



- Czas wykonania $T(2, n) = 5$
- Przyspieszenie $S(2, n) = \frac{T(1,n)}{T(2,n)} = \frac{8}{5}$
- Efektywność $E(2, n) = \frac{S(2,n)}{2} = \frac{8/5}{2} = \frac{8}{10}$
- Koszt $C(p, n) = p \cdot T(p, n) = 2 \cdot T(2, n) = 10$



- Czas wykonania $T(6, n) = 3$
- Przyspieszenie $S(6, n) = \frac{T(1,n)}{T(6,n)} = \frac{8}{3}$
- Efektywność $E(6, n) = \frac{S(6,n)}{6} = \frac{8/3}{6} = \frac{8}{18} \approx 44\%$
- Koszt $C(p, n) = p \cdot T(p, n) = 6 \cdot T(6, n) = 18$

Prawo Amdahla

Maksymalne wartość przyspieszenia określona jest wzorem podanym przez Gena Amdahla

$$S(p,n) = \frac{1}{s + (1-s)/p}$$

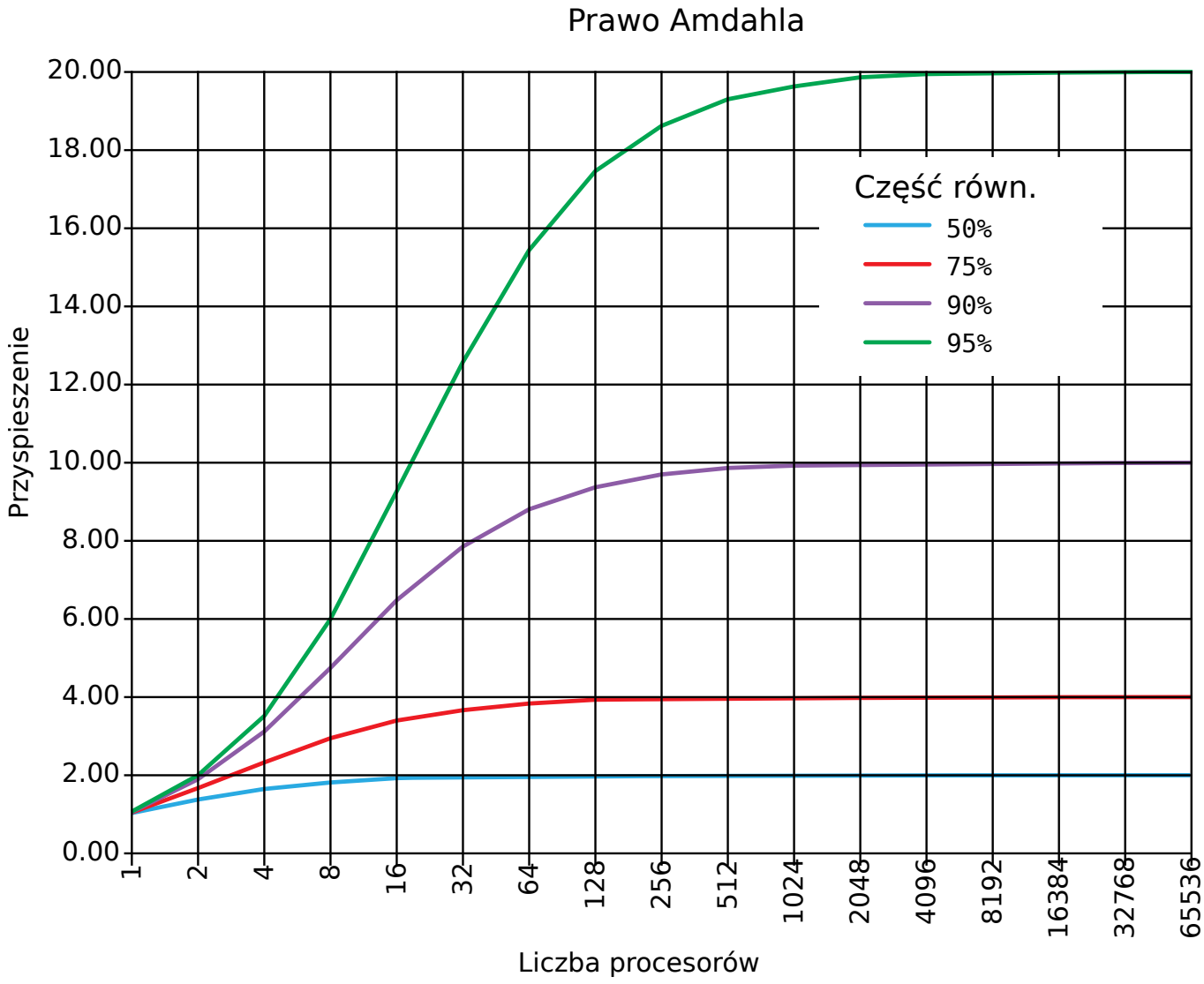
sekwencyjna część obliczeń

równoległa część obliczeń

- s – część obliczeń w algorytmie, które muszą być wykonane sekwencyjnie
- p – liczba procesorów

Rysunek : Gene Amdahl (2008)

Prawo Amdahla



Prawo Amdahla – c.d.

- Prawo Amdahla podaje również górne ograniczenie przyspieszenia

$$\lim_{p\to\infty} \frac{1}{s+(1-s)/p} = \frac{1}{s}$$

- Przykładowo, dla części sekwencyjnej stanowiącej 1% obliczeń w algorytmie ($s = 0.01$) górna wartość przyspieszenia wynosi 100.
- W rzeczywistości sytuacja wygląda nawet gorzej, ponieważ w prawie Amdahla przyjęto założenia upraszczające:
  - obliczenia równoległe dają się idealnie równo podzielić między procesorami
  - pominięto czas na organizację obliczeń równoległych, komunikację itd. ($T^o(p,n)$)

Prawo Amdahla – uwagi

- W prawie Amdahla celem było skrócenie czasu obliczeń przy zadanym, stałym rozmiarze danych wejściowych (i stałym czasie dla części sekwencyjnej s)
- Można jednak podejść do problemu od innej strony – zachować stały czas części równoległej, a zwiększać rozmiar danych wejściowych, czyli n
- Przykładowo, symulację meteorologiczną można wykonać na 100 procesorowym komputerze w czasie 1 godziny z dokładnością siatki 1km
  - na komputerze o 10000 procesorach można tę samą symulację wykonać w czasie 1 godziny, ale z dokładnością siatki do 100m

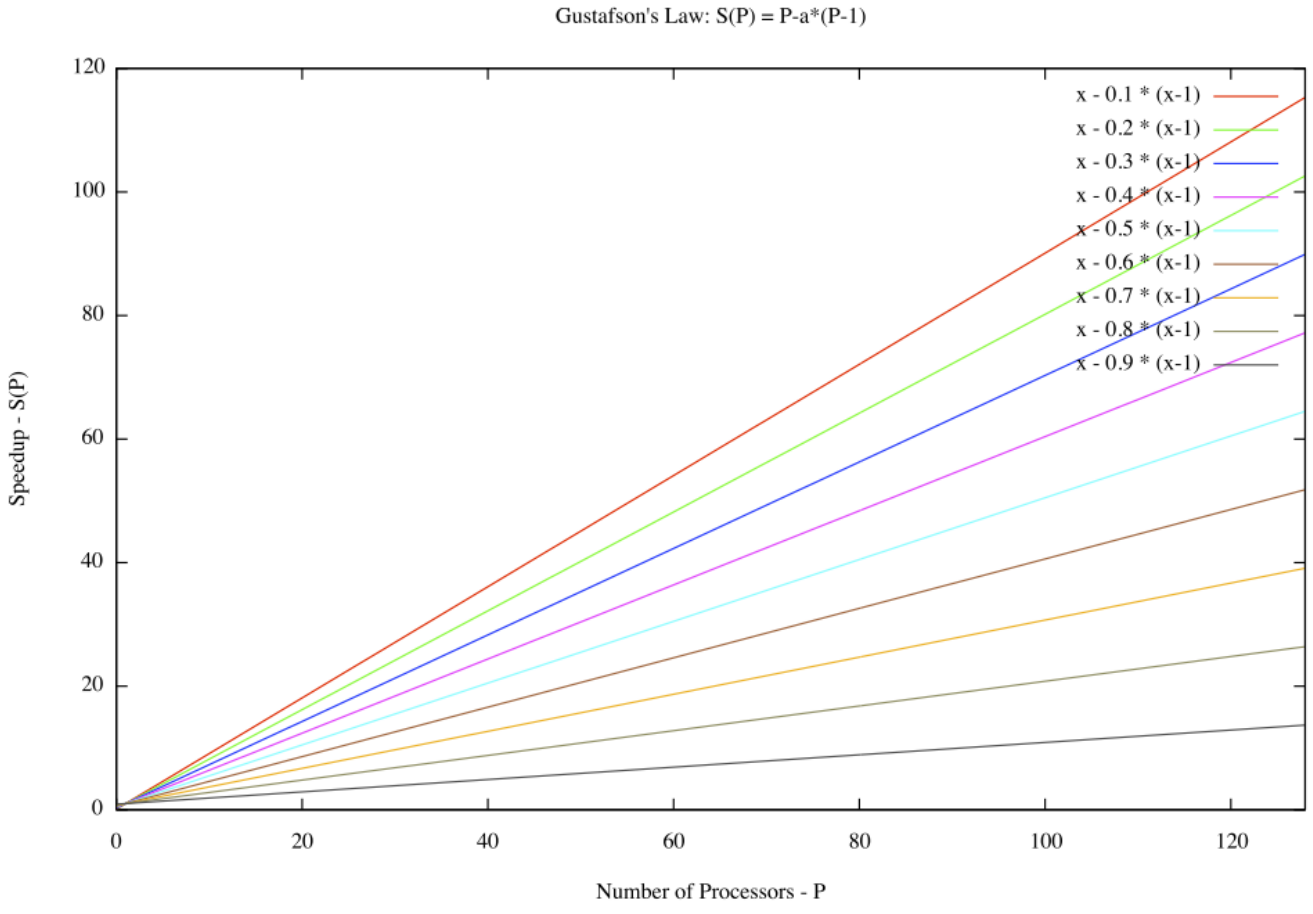
Prawo Gustafsona i Barsisa

- Niech:
  - p – liczba procesorów
  - σ – część czasu obliczeń w algorytmie równoległym przypadająca na część sekwencyjną
  - ρ – część czasu obliczeń wykonywana równolegle (pomijamy czas na komunikację itp.)
- Oczywiście, $\sigma + \rho = 1$
- Jeżeli algorytm ten wykonany zostałby na komputerze sekwencyjnym, to czas jego obliczeń wyniósłby: $\sigma + p\rho$
- Wynika stąd, że przyspieszenie wyniosłoby:

$$\Psi(p, n) \leqslant \frac{\sigma + p\rho}{\sigma + \rho} = \sigma + p\rho = \sigma + p(1 - \sigma) = p + (1 - p)\sigma$$

Powyższy wzór znany jest jako **prawo Gustafsona i Barsisa**

Prawo Gustafsona i Barsisa



Rysunek : Przyspieszenie w zależności od części sekwencyjnej obliczeń

Prawo Gustafsona i Barsisa

- Prawo Gustafsona i Barsisa określa skalowalne przyspieszenie, ponieważ wraz ze wzrostem liczby procesorów rośnie rozmiar problemu, tak aby zachować stały czas obliczeń równoległych
- W prawie Amdahla rozmiar problemu pozostawał stały, a czas obliczeń równoległych ulegał skróceniu
- Prawo Gustafsona i Barsisa pozwala na osiągnięcie większych przyspieszeń, np. dla części sekwencyjnej równej 1% ($\sigma = 0.01$) oraz $p = 1000$ przyspieszenie $\Psi(p, n) \approx 990$
- Problem: wraz ze wzrostem rozmiaru danych wejściowych rośnie zapotrzebowanie na pamięć operacyjną, dyskową, przepustowość sieci

Uwagi

- W celu uzyskania dużych przyspieszeń konieczna jest nie tylko duża część obliczeń, którą można zrównoleglić
- Zarówno prawo Amdahla, jak i prawo Gustafsona pomijają czas $T^o(p, n)$ poświęcony na wymianę danych między procesorami oraz organizację obliczeń równoległych
- Szczególnie w komputerach z rozproszoną pamięcią wymiana komunikatów jest kosztowna – należy ograniczać komunikację
- Rzadka komunikacja zazwyczaj wymaga podziału problemu na większe części, tak by nie zachodziła konieczność częstego pobierania kolejnej „porcji" danych
- Oba prawa zakładają równy podział obliczeń między procesorami, co w wielu przypadkach jest trudne do uzyskania

Uwagi

Ułatwiony dostęp do komputerów z wieloma jednostkami obliczeniowymi nie zwalnia nas z konieczności uważnego doboru odpowiedniego do zadania algorytmu.

Rozmiar danych (n)	Alg. Sekw. $T(n) = n \log n$	Alg. Równ. $T(p,n) = \frac{n^2}{p}$		
		p=10	p=1000	p=100000
1000	$1 \cdot 10^{-5}$ sek	0,0001 sek	0,000001 sek	0,00000001 sek
1000000	0,02 sek	100 sek	1 sek	0,01 sek
1000000000	29,9 sek	3,17 lat	11,57 dni	2,78 godz.