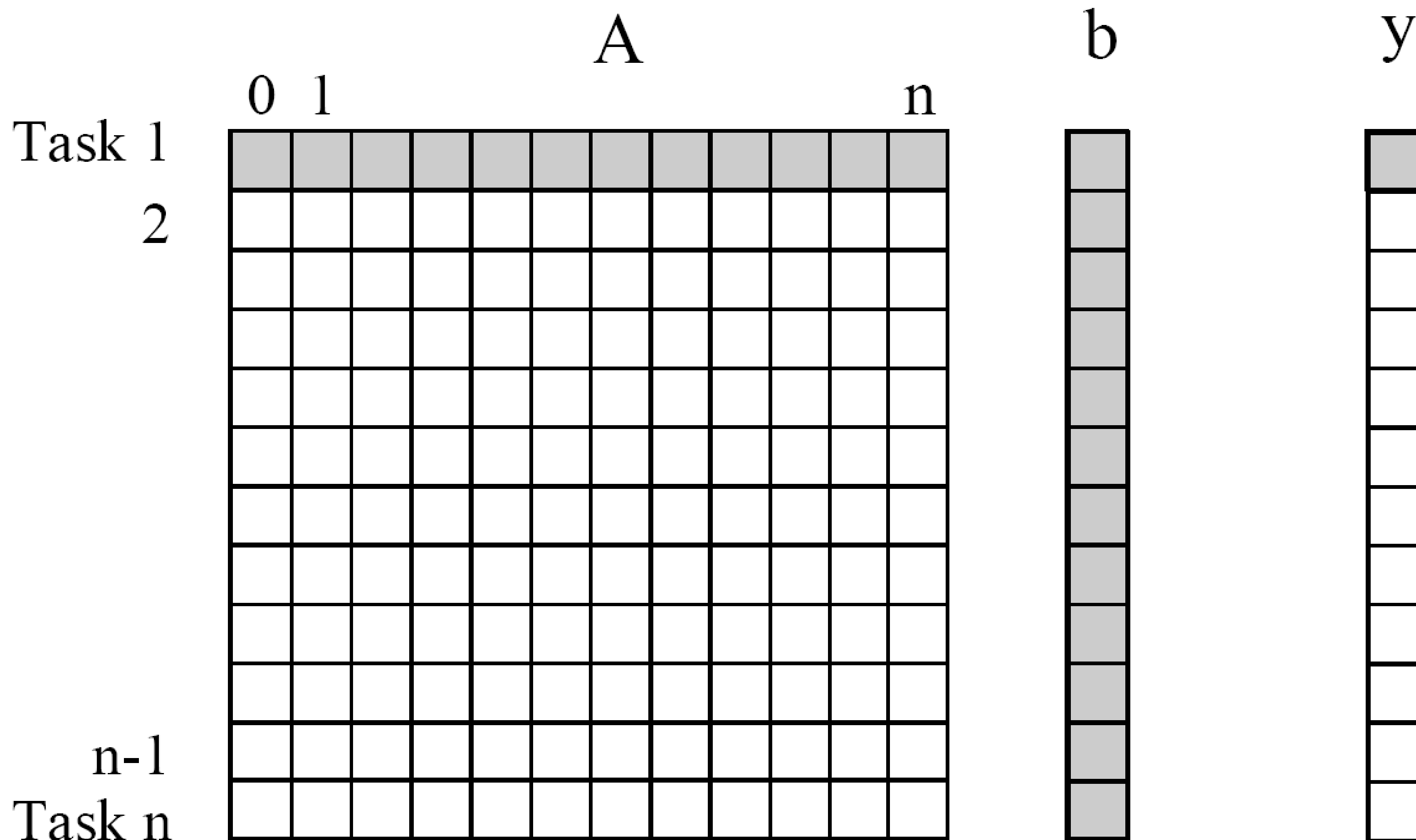
Jak zrównolegla się algorytmy - projektowanie algorytmów równoległych

Dekompozycja problemu

- Pierwszym krokiem w zrównoleglaniu algorytmu jest podział problemu na zadania (ang. task), które mogą być wykonane niezależnie. Podział ten nazywany jest ***dekompozycją***.
- Problem może być podzielony na zadania na wiele różnych sposobów.
- Zadania mogą być takich samych bądź też różnych rozmiarów.
- Podział zbioru na zadania może być zilustrowany w postaci grafu skierowanego, którego wierzchołki odpowiadają zadaniom, a krawędzie wskazują, że wynik jednego z zadań jest potrzebny przez drugie.
- Graf ten nazywamy grafem zależności zadań (ang. task dependency graph.)

Przykład: mnożenie macierzy i wektora



- y=Ab, gdzie b oraz y to n-elementowe wektory, A jest macierzą nxn.
- Przypomnienie i-ty element wektora y obliczamy mnożąc i-ty wiersz macierzy A przez wektor b.
- Jedno zadanie = obliczenie jednego elementu y.

Mnożenie macierzy i wektora

- Obliczenie jednego elementu n-elementowego wektora y, będącego iloczynem macierzy nxn A i wektora n-elementowego b, jest niezależne od obliczenia pozostałych elementów y.
- Wykorzystując ten fakt możemy podzielić problem na n zadań.
- Pomimo, że zadania współdzielą dane (wektor b), to nie ma pomiędzy nimi żadnych zależności sterujących. Żadne z zadań nie potrzebuje wyniku jakiegokolwiek z pozostałych. Oznacza to, że żądne z zadań nie musi czekać na wykonanie pozostałych.
- Graf zależności zadań składa się z n wierzchołków i nie ma krawędzi.
- Każde zadanie ma taki sam rozmiar, jeżeli chodzi o liczbę elementarnych operacji arytmetycznych.
- Pytanie: czy problemu nie udałoby się podzielić na więcej zadań ?

Przykład: zapytanie do bazy danych

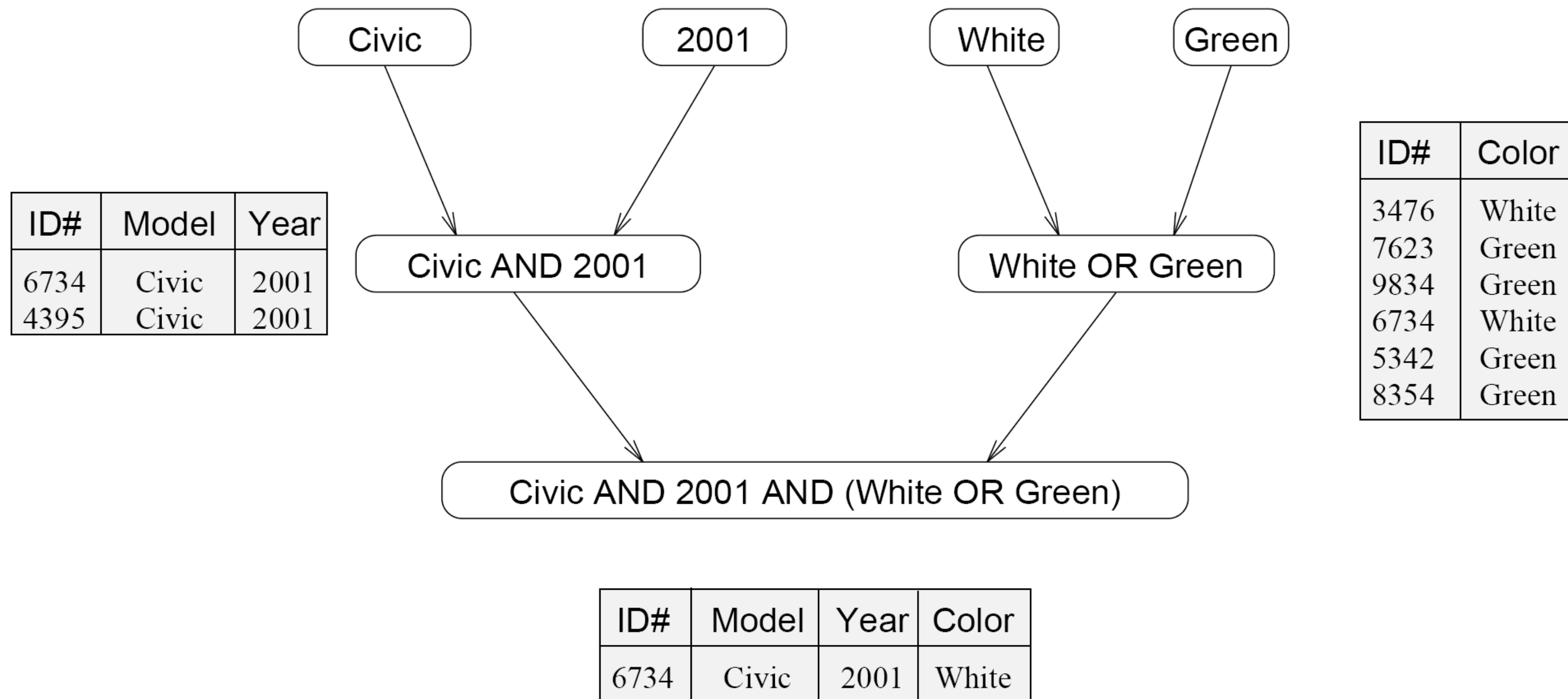
- Rozważmy zapytanie:

  MODEL=”CIVIC” AND YEAR=2001 AND (COLOR=”GREEN” OR COLOR=”WHITE”)

skierowane do następującej tabeli w bazie danych:

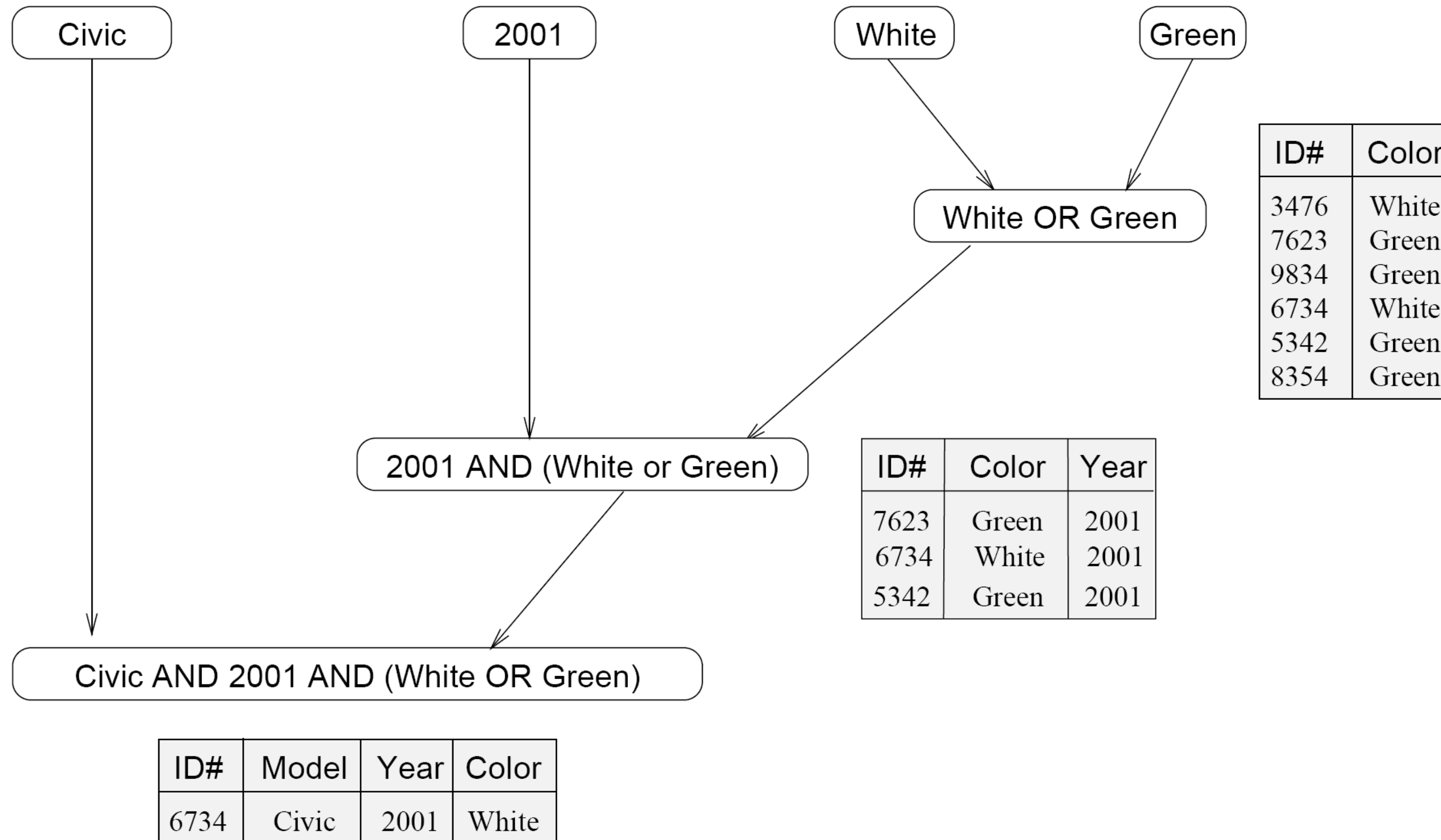
ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

Zapytanie do bazy danych - graf zadań



- Tym razem w grafie zadań występują zależności. Niektóre zadania wymagają przed uruchomieniem wyniku niektórych pozostałych. Pomimo, że jest 7 zadań, to nie możemy ich uruchomić równolegle.

Graf zadań - alternatywa



ID#	Model
4523	Civic
6734	Civic
4395	Civic
7352	Civic

ID#	Year
7623	2001
6734	2001
5342	2001
3845	2001
4395	2001

ID#	Color
3476	White
6734	White

ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

ID#	Color
3476	White
7623	Green
9834	Green
6734	White
5342	Green
8354	Green

ID#	Color	Year
7623	Green	2001
6734	White	2001
5342	Green	2001

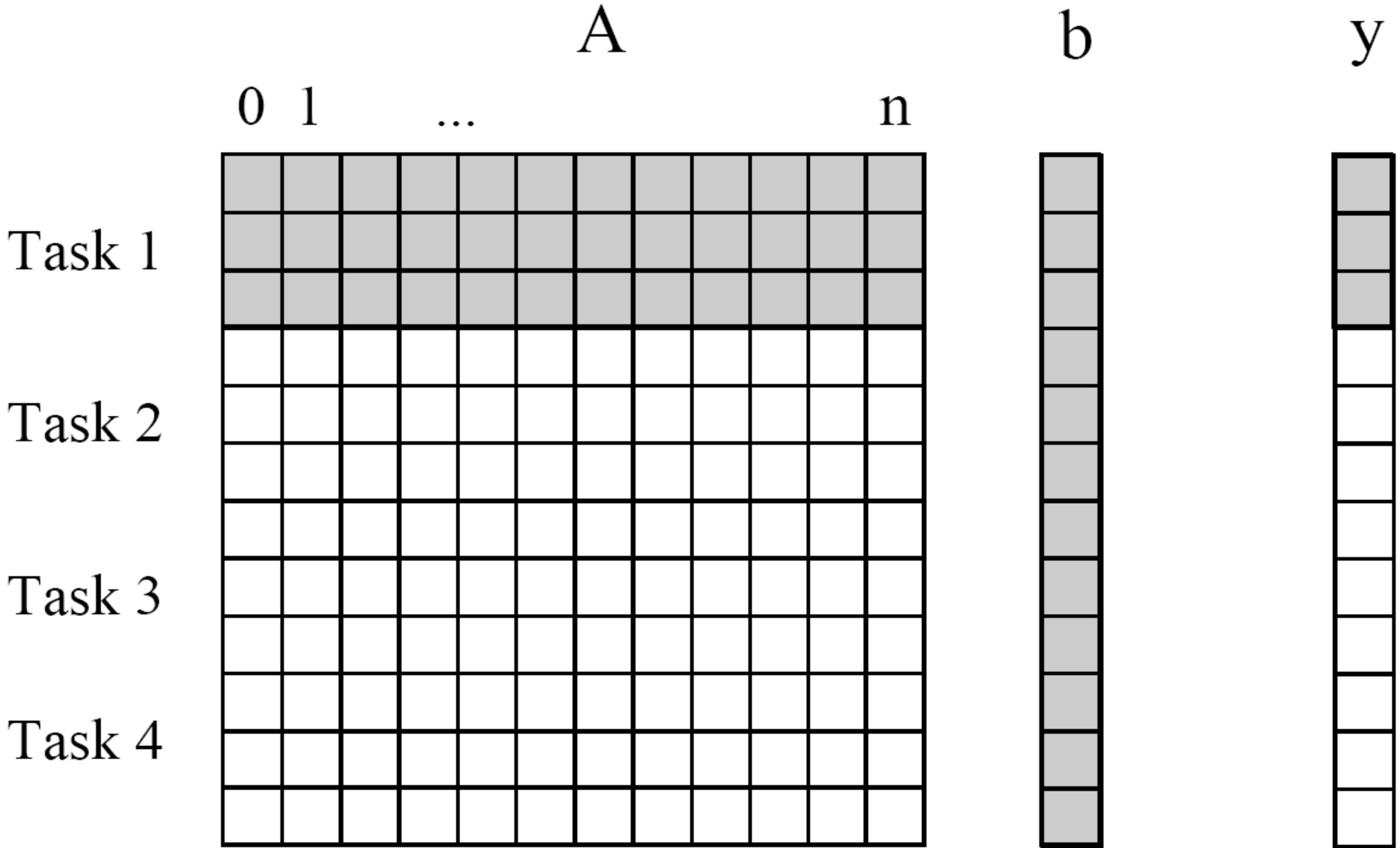
ID#	Model	Year	Color
6734	Civic	2001	White

- Obserwacja: Ten sam problem można podzielić na zadania na różny sposób, co może prowadzić do dużych różnic w wydajności algorytmu równoległego.

Ziarnistość (ang. granularity) dekompozycji i stopień współbieżności

- Liczba zadań na które dzielimy problem określają nam ziarnistość dekompozycji.
  - Dekompozycja gruboziarnista (ang. coarse grained) - problem podzielony na niewielką liczbę stosunkowo dużych zadań.
  - Dekompozycja drobnoziarnista (ang. fine grained) - problem podzielony na dużą liczbę niewielkich zadań.
- Stopień współbieżności (ang. degree of concurency) - jest liczbą zadań, które mogą być wykonywane równolegle.
  - Może się on zmieniać w trakcie wykonywania programu, stąd wyrózniamy maksymalny stopień współbieżności oraz średni stopień współbieżności.
- Stopień współbieżności zwiększa się, gdy dekompozycja staje się bardziej drobnoziarnista i na odwrót.
  - Ale nie oznacza to, że dekompozycja drobnoziarnista prowadzi do lepszej wydajności. Zwiększanie ziarnistości prowadzi bowiem do większej interakcji pomiędzy zadaniami, co zwiększa czas wykonania programu (koszty komunikacji).

Mnożenie macierz*wektor - zmniejszenie ziarnistości

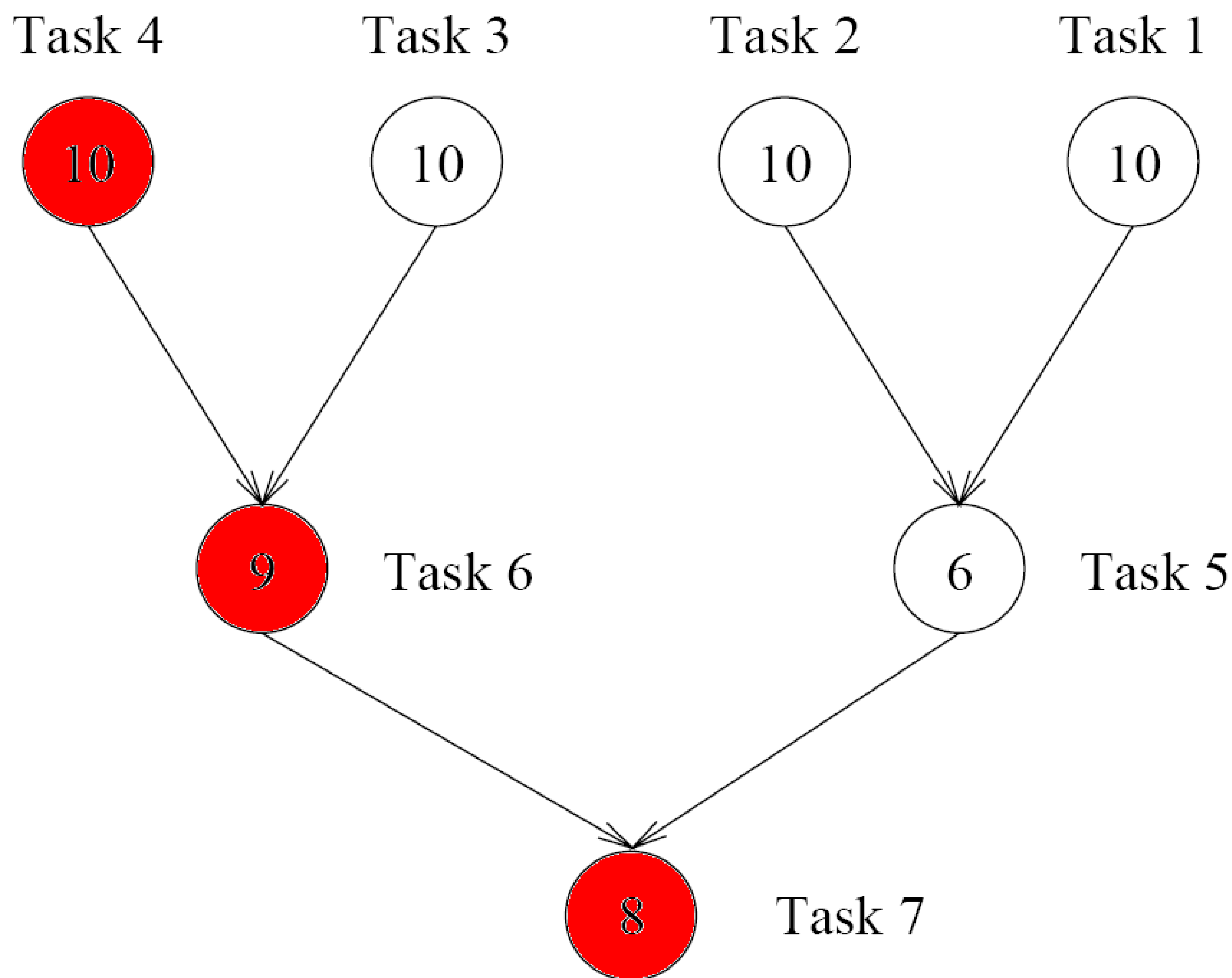


- Jedno zadanie zajmuje się obliczeniem trzech (a nie jednego, jak w poprzednim przykładzie) elementów wektora y.

Długość ścieżki krytycznej

- Ścieżka w grafie zależności zadań reprezentuje ciąg czynności, które muszą być wykonane jedna po drugiej.
- Nadajmy każdemu wierzchołkowi wagę określającą czas wykonania danego zadania.
- Wierzchołki bez krawędzi przychodzącej nazwiemy ***wierzchołkami początkowymi***, a wierzchołki bez krawędzi wychodzącej ***wierzchołkami końcowymi***.
- Długość ścieżki jest sumą wag wierzchołków na niej leżących.
  - ***Długość ścieżki krytycznej*** (ang. critical path length) maksymalna długość ścieżki pomiędzy wierzchołkiem początkowym a końcowym w grafie zależności zadań.
- Długość ścieżki krytycznej określa maksymalny czas pracy programu równoległego.
- Średni stopień współbieżności możemy wyliczyć jako stosunek sumy wag wszystkich wierzchołków do długości ścieżki krytycznej.

Długość ścieżki krytycznej - przykład



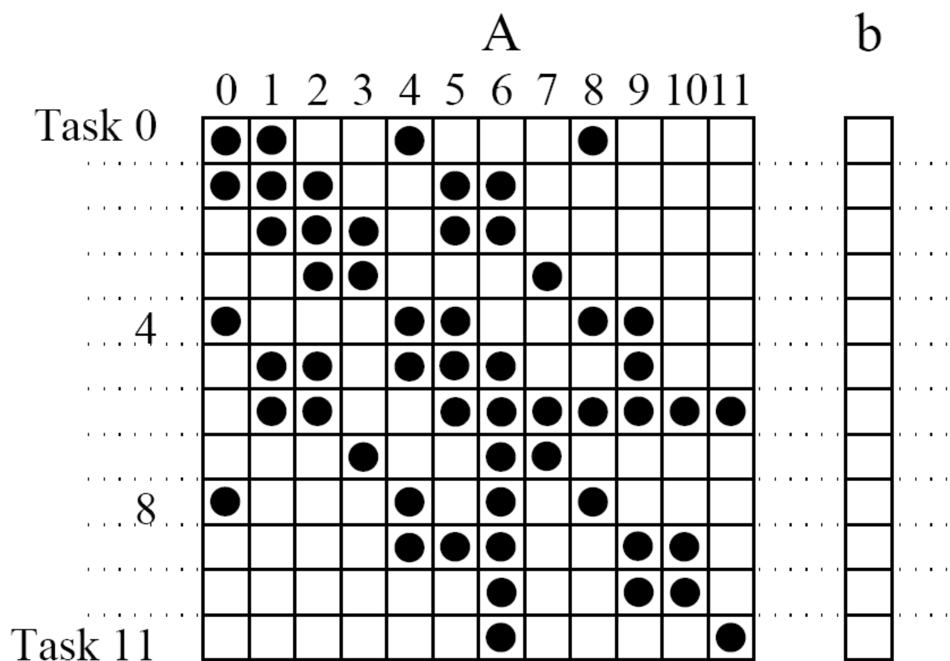
- Długość ścieżki krytycznej: 27
- Maksymalny stopień współbieżności: 4
- Średni stopień współbieżności: 2.33

Ograniczenia w zrównoleglaniu

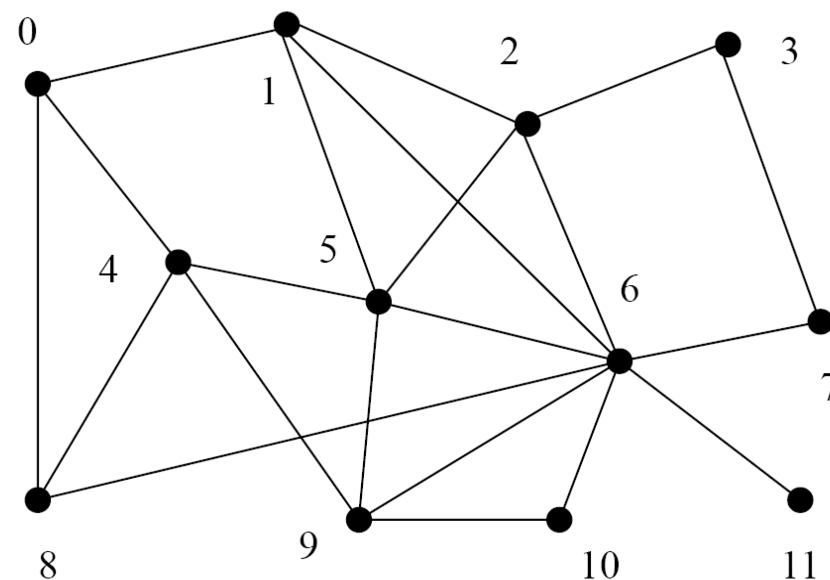
- Z dotychczasowego opisu wydaje się, że dysponując dowolnie dużą liczbą procesorów możemy, zwiększając ziarnistość dekompozycji, dowolnie zmniejszyć czas pracy programu.
- Niestety zawsze mamy do czynienia z ograniczeniem liczby zadań na które możemy zdekomponować problem.
- Na przykład przy mnożeniu macierzy nxn i wektora n-elementowego maksymalna liczba zadań, na które możemy podzielić problem, jest równa $O(n^2)$.
  - Ponieważ wykonanie mnożenia wymaga n^2 mnożeń i dodawań.
- Kolejnym ograniczeniem jest jest istnienie ***interakcji*** pomiędzy zadaniami wykonywującymi się na różnych procesorach.
  - Zadania mogą współdzielić dane wejściowy, wyjściowe lub pośrednie.
  - Zależności pomiędzy zadaniami zawsze prowadzą do interakcji: wyniki jednego zadania są danymi wejściowymi drugiego.
- Interakcje mogą istnieć pomiędzy niezależnymi (w grafie zależności) zadaniami
  - Na przykład przy mnożeniu macierzy i wektora wszystkie zadania potrzebują kopii wektora b. Ponieważ na początku jest tylko jedna kopia, każde zadanie musi ją uzyskać co może prowadzić do interakcji.

Graf interakcji zadań

- Jest to graf niezorientowany, w którym ścieżka pomiędzy dwoma wierzchołkami reprezentującymi dwa zadania oznacza istnienie interakcji. Graf zależności zadań jest z reguły podgrafem grafu interakcji.
- Przykład: mnożenie macierzy A rzadkiej i wektora b z podziałem wektora b pomiędzy zadania. Każde zadanie odpowiada za jeden element b i wysyła go pozostałym.



macierz rzadka
elementy ze znakiem ●
są różne od zera

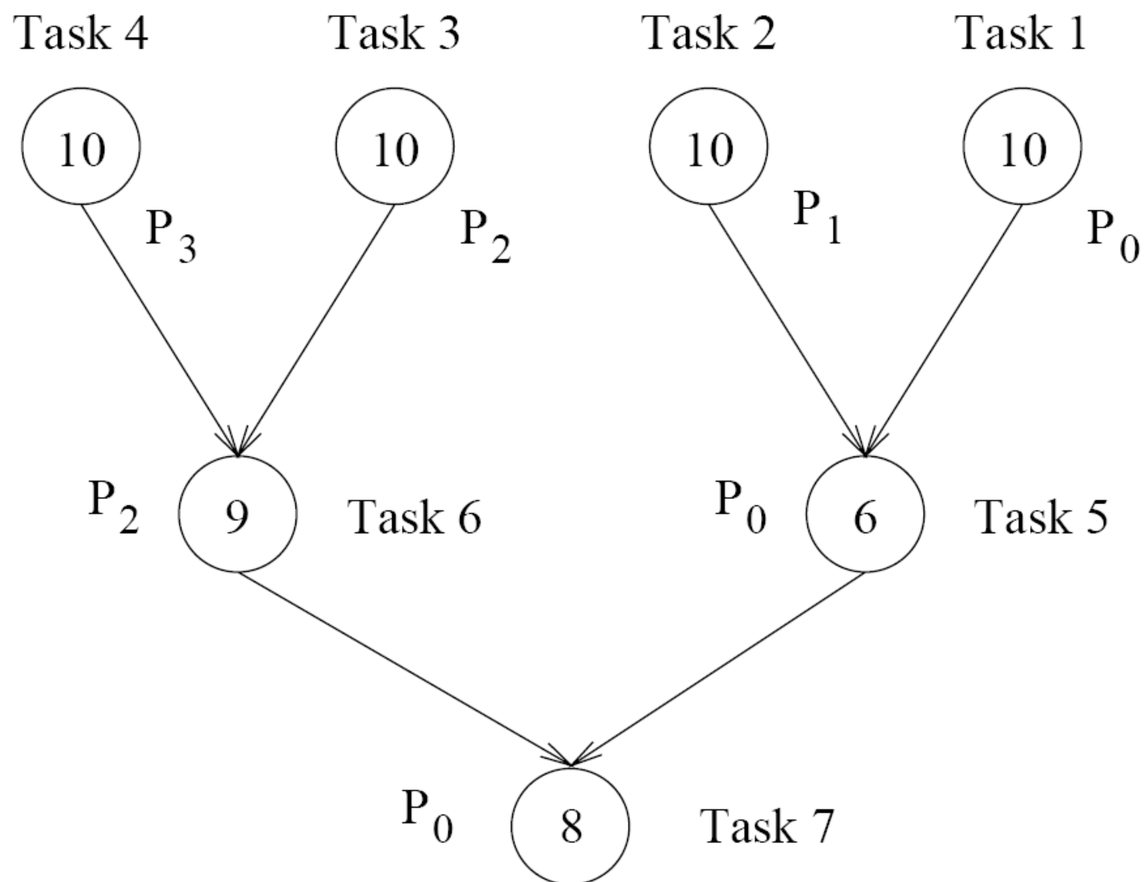


graf interakcji zadań

Alokacja zadań procesom (procesorom)

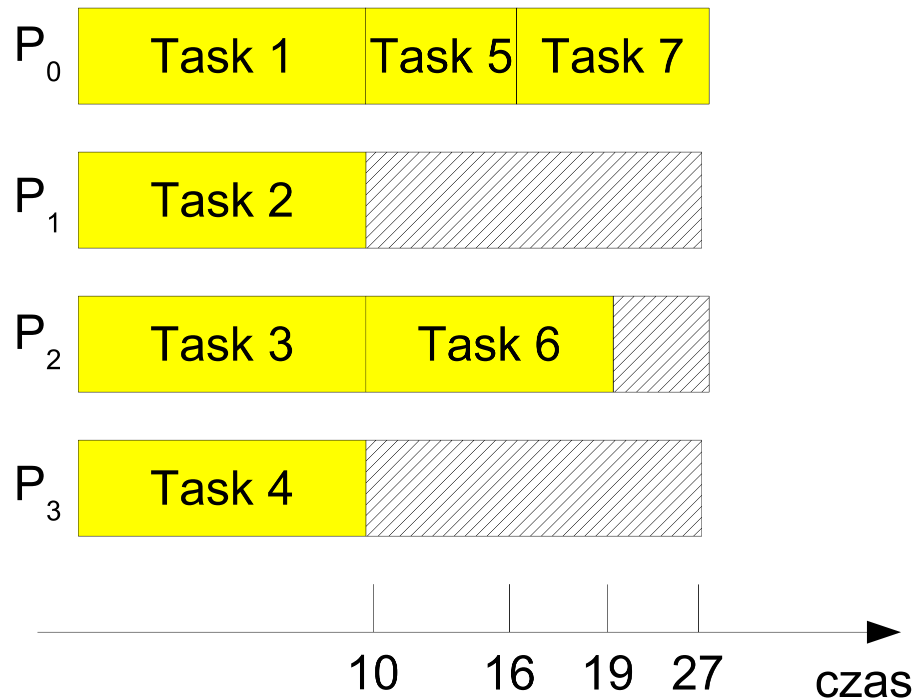
- Na potrzeby tego wykładu przyjmujemy odpowiedniość jeden proces=jeden procesor. W dalszej części będziemy mówić raczej o procesach, ponieważ typowe interfejsy programistyczne np Uniks nie pozwalają na umieszczenia zadań na fizycznych procesorach.
  - Polegamy na systemie operacyjnym, aby odwzorował procesy na procesory.
- Na ogół liczba zadań jest znacznie większa niż liczba dostępnych procesorów. Z tego powodu musimy skonstruować odwzorowanie zadań w procesy.
- Konstruując to odwzorowanie (ang. mapping) bierzemy pod uwagę zarówno grafy zależności jak i interakcji zadań.
  - Bierzemy pod uwagę graf zależności, aby starać się zapewnić że proces nie będzie oczekiwał bezczynnie na dane i że problem jest idealnie podzielony na procesy (zrównoważenie obciążenia - ang. load balance)
  - Bierzemy pod uwagę graf interakcji aby zminimalizować komunikację pomiędzy procesami.
- Alokacja zadań procesom ma decydujący wpływ na wydajność algorytmu równoległego.

Alokacja zadań: przykład



- Proces P_0: zadania 1,5,7; proces P_1: zadanie 2; proces P_2 zadania 3,6; proces P_3: zadanie 4.

Przykład alokacji zadań - diagram Gannt'a



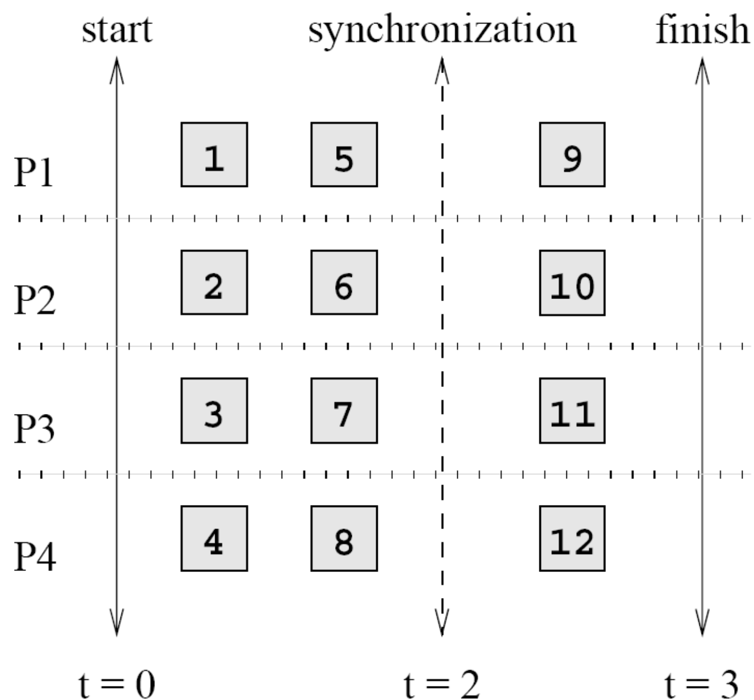
Obszar zakreskowany => bezczynność procesu

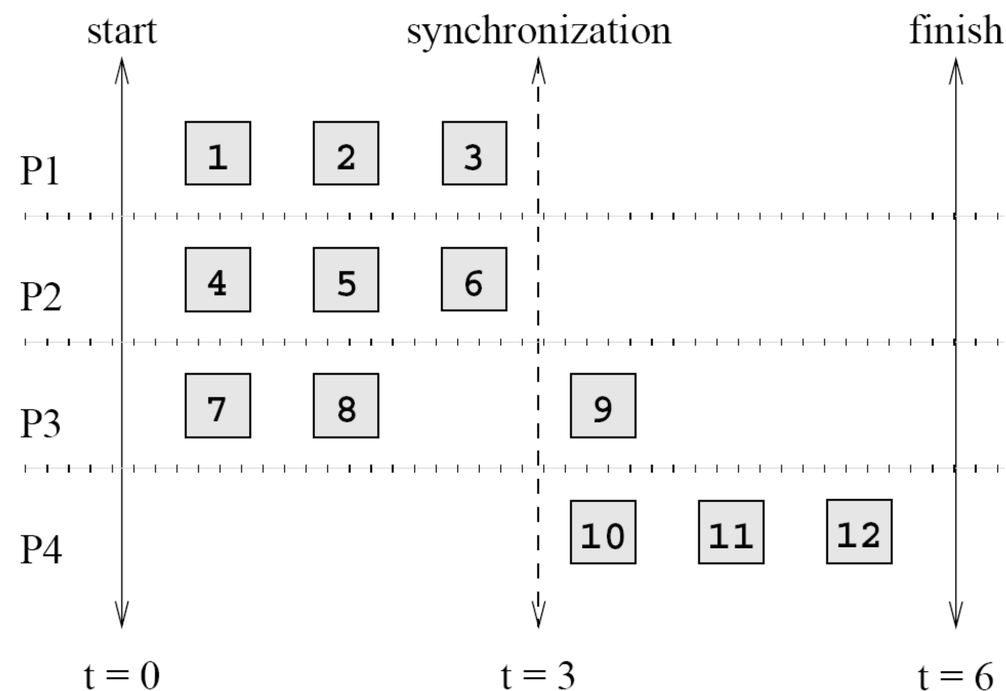
- Pomimo wykorzystania czterech procesów nie osiągniemy przyspieszenia równego cztery, a 2.33
- Obciążenie nie jest idealnie zrównoważone

Alokacja zadań - kryteria

- Wybierając alokację zadań bierzemy pod uwagę następujące kryteria:
  - odwzorowanie niezależnych zadań w różne procesy aby zmaksymalizować przyspieszenie.
  - zbalansowane obciążenie procesów.
  - jak najwcześniejsze odwzorowanie zadań na ścieżce krytycznej.
  - minimalizację interakcji pomiędzy procesami, poprzez przypisanie zadań o gęstym profilu interakcji tym samym procesom.
- Niestety te kryteria są w konflikcie ze sobą. Na przykład umieszczenie wszystkich zadań w jednym procesie prowadzi do zerowego stopnia interakcji międzyprocesowych. W takim przypadku nie uzyskamy jednak przyspieszenia obliczeń.
- Generalnie problem alokacji zadań procesom jest NP-zupełny.

Alokacja zadań - pułapki





- Przykład w którym zadania 9-12 mogą się rozpocząć dopiero po wykonaniu zadań 1-8 (linia synchronization).
- W obydwu alokacjach zadania podzielono równomiernie na procesy, jednak alokacja po lewej stronie jest korzystniejsza
- Nie zawsze równomierne podzielenie (na podstawie wymagań obliczeniowych) zadań na procesy prowadzi do wyrównania obciążenia.

Alokacja zadań - metody

- Metody alokacji zadań możemy podzielić na statyczne i dynamiczne.
- Metody statyczne alokują zadania procesom *a priori* przed uruchomieniem algorytmu.
  - Podział na zadania musi być znany przed uruchomieniem algorytmu.
  - Musimy znać rozmiar każdego zadania przed uruchomieniem algorytmu.
- Metody statyczne oparte są na technikach optymalizacji dyskretnej (np. podział grafu zależności/lub interakcji zadań).
- Metody dynamiczne alokują zadania podczas pracy programu. Mają zastosowanie do zadań które:
  - są generowane dynamiczne.
  - nie są znane ich rozmiary.
- Przykładem dynamicznej alokacji zadań był program z wykładu czwartego obliczający zbiór Mandelbrota.
  - Wykorzystywał model master-slave z procesem zarządcą rozdzielającym zadania

Techniki dekompozycji danych

- W jaki sposób dekomponujemy problem na zadania ?
- Niestety nie ma jednej recepty, która zawsze skutkuje. Na wykładzie zostaną zaprezentowane najpowszechniejsze techniki, mające zastosowanie w szerokiej klasie problemów. Są to:
- Dekompozycja rekurencyjna.
- Dekompozycja danych
- Dekompozycja eksploracyjna
- Dekompozycja spekulacyjna.
- Dekompozycja hybrydowa, będąca połączeniem niektórych spośród czterech poprzednich.

Dekompozycja rekurencyjna

- Jest dobra dla problemów, które mogą być rozwiązane przy pomocy paradygmatu „dziel i zwyciężaj” (ang. divide and conquer).
- Przypomnienie: funkcja rekurencyjna, to taka funkcja która wywołuje samą siebie.
- Problem jest dekomponowany na podbroblemy o mniejszym wymiarze.
  - Podproblemy są niezależne od siebie i mogą być rozwiązane równolegle.
- Rozwiązanie problemu wykorzystuje rozwiązania podproblemów.
- Podproblemy są rekurencyjnnie dekomponowane dalej, aż do osiągnięcia wymaganej ziarnistości.
- Po zakończeniu dekompozycji podbroblemy stają się zadaniami.
- Nawet jeżeli powszechnie używany algorytm nie jest oparty na strategii „dziel i zwyciężaj” często możemy go przeformułować tak aby poddawał się łatwo dekompozycji rekurencyjnej.
  - przykładem jest znajdowanie minimum w tablicy.

Przykład: znajdowanie minimum

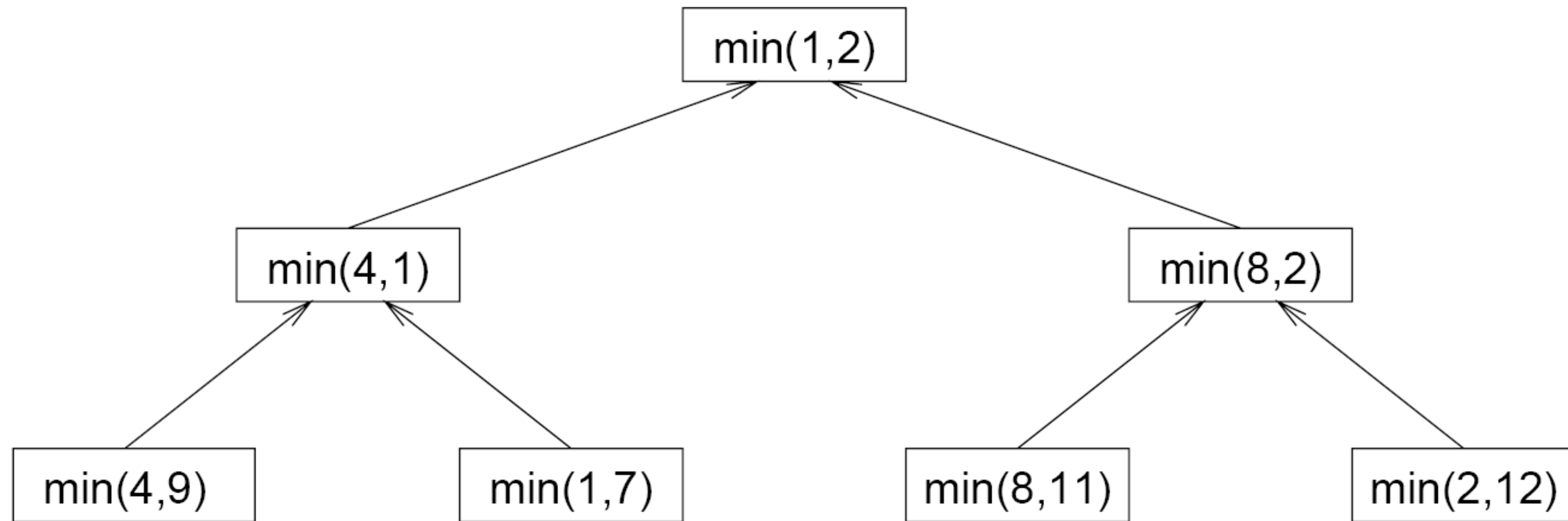
```
1.    procedure SERIAL_MIN (A, n)
2.    begin
3.    min = A[0];
4.    for i := 1 to n − 1 do
5.        if (A[i] < min) min := A[i];
6.    endfor;
7.    return min;
8.    end SERIAL_MIN
```

wersja iteracyjna

```
1.     procedure RECURSIVE_MIN (A, n)
2.     begin
3.     if (n = 1) then
4.         min := A[0];
5.     else
6.         lmin := RECURSIVE_MIN (A, n/2);
7.         rmin := RECURSIVE_MIN (&(A[n/2]), n − n/2);
8.         if  (lmin < rmin)  then
9.             min := lmin;
10.        else
11.            min := rmin;
12.        endelse;
13.    endelse;
14.    return min;
15.    end RECURSIVE_MIN
```

wersja rekurencyjna
zasada „dziel i zwyciężaj”

Znajdowanie minimum - graf zależności zadań



- Szukamy minimum dla ciągu {4,9,1,7,8,11,2,12}
- Każde zadanie oblicza minimum z pary.

Dekompozycja danych

- Jest to technika powszechnie używana do problemów operujących na dużych ilościach danych.
- Dekompozycja wykonywana jest w dwóch krokach.
  - w kroku pierwszym dzielone są dane, na których przeprowadzane są obliczenia.
  - w kroku drugim ten podział danych wykorzystywany jest do podziału obliczeń.
- Podział danych może być wykorzystany na wiele sposobów:
  - podział danych wyjściowych (wyników). Jest możliwy do wykonania jeżeli element wyników może być obliczony niezależnie od pozostałych jako funkcja danych wejściowych.
  - podział danych wejściowych
  - podział danych pośrednich.
  - dowolna kombinacja powyższych trzech metod.

Dekompozycja wyników - mnożenie macierzy

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

- Każda z macierzy nxn została podzielona na cztery macierze o wymiarze n/2xn/2.
- Każdą podmacierz macierzy wyjściowej C możemy obliczyć niezależnie, co prowadzi do dekompozycji na cztery zadania:

$$\begin{aligned}
\text{Task 1: } C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\
\text{Task 2: } C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\
\text{Task 3: } C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\
\text{Task 4: } C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2}
\end{aligned}$$

- W powyższym przykładzie mnożenie i dodawanie są operacjami macierzowymi !

Podział danych pośrednich

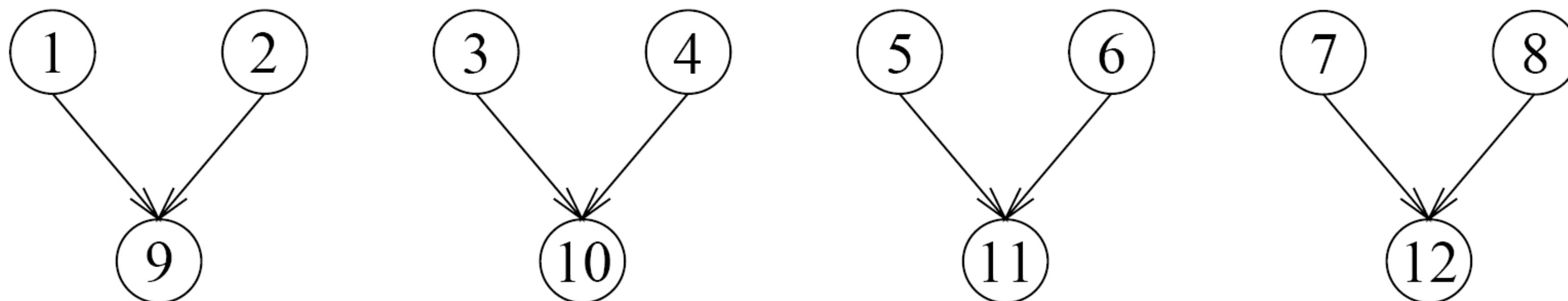
- W poprzednim przykładzie $C_{1,1}$ było obliczone jako:

$$C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$$

- Obliczenie dwóch składników sumy można zdekomponować dalej, wykonując równolegle dwa iloczyny $A_{1,1}B_{1,1}$ oraz $A_{1,2}B_{2,1}$. Otrzymamy dekompozycję:

$$\begin{array}{ll|ll}
\text{Task 01:} & D_{1,1,1} = A_{1,1}B_{1,1} & & \\
\text{Task 02:} & D_{2,1,1} = A_{1,2}B_{2,1} & & \\
\text{Task 03:} & D_{1,1,2} = A_{1,1}B_{1,2} & \text{Task 09:} & C_{1,1} = D_{1,1,1} + D_{2,1,1} \\
\text{Task 04:} & D_{2,1,2} = A_{1,2}B_{2,2} & \text{Task 10:} & C_{1,2} = D_{1,1,2} + D_{2,1,2} \\
\text{Task 05:} & D_{1,2,1} = A_{2,1}B_{1,1} & \text{Task 11:} & C_{2,1} = D_{1,2,1} + D_{2,2,1} \\
\text{Task 06:} & D_{2,2,1} = A_{2,2}B_{2,1} & \text{Task 12:} & C_{2,2} = D_{1,2,2} + D_{2,2,2} \\
\text{Task 07:} & D_{1,2,2} = A_{2,1}B_{1,2} & & \\
\text{Task 08:} & D_{2,2,2} = A_{2,2}B_{2,2} & &
\end{array}$$

Podział danych pośrednich - graf zależności zadań



- Pomimo istnienia 12 zadań nie otrzymamy 12-krotnego przyspieszenia, ponieważ niektóre zadania czekają na wyniki innych.

Dekompozycja danych wejściowych - znajdowanie minimum

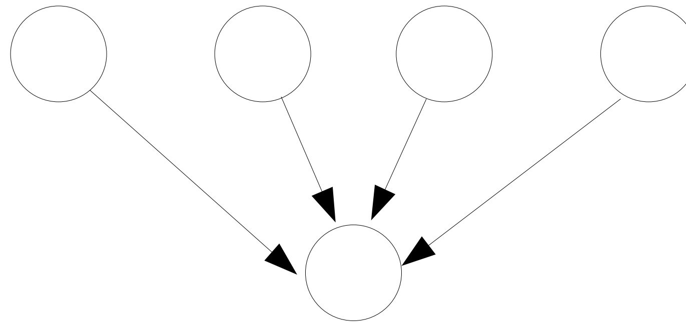
3	7	2	9

11	4	5	8

7	10	6	13

19	1	3	9

- Problem znajdowania minimum w tablicy. Jedno zadanie znajduje minimum w swojej części tablicy. Następnie ostatnie zadanie oblicza minimum z czterech minimów.
- Otrzymujemy następujący graf zależności:



- Generalnie: z każdą partycją podzielonego zbioru danych wejściowych związane jest jedno zadanie. Wykonuje ono tak dużo obliczeń na swoich lokalnych danych jak jest to możliwe. Dalsze przetwarzanie wykorzystuje wyniki tych zadań.

Dekompozycja eksploracyjna

- Dekompozycja eksploracyjna jest wykorzystywana w przypadku problemów, których rozwiązanie wymaga przeszukania pewnej przestrzeni rozwiązań (ang. solution space).
  - Problemy optymalizacji dyskretnej, dowodzenia twierdzeń, rozgrywania gier.
- W dekompozycji eksploracyjnej dzielimy przestrzeń rozwiązań na części i przeszukujemy niezależnie każdą z tych części do momentu znalezienia rozwiązania.
- Przykład 15-układanka (15 - puzzle)

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

konfiguracja początkowa

1	2	3	4
5	6	7	8
9	10		11
13	14	15	12

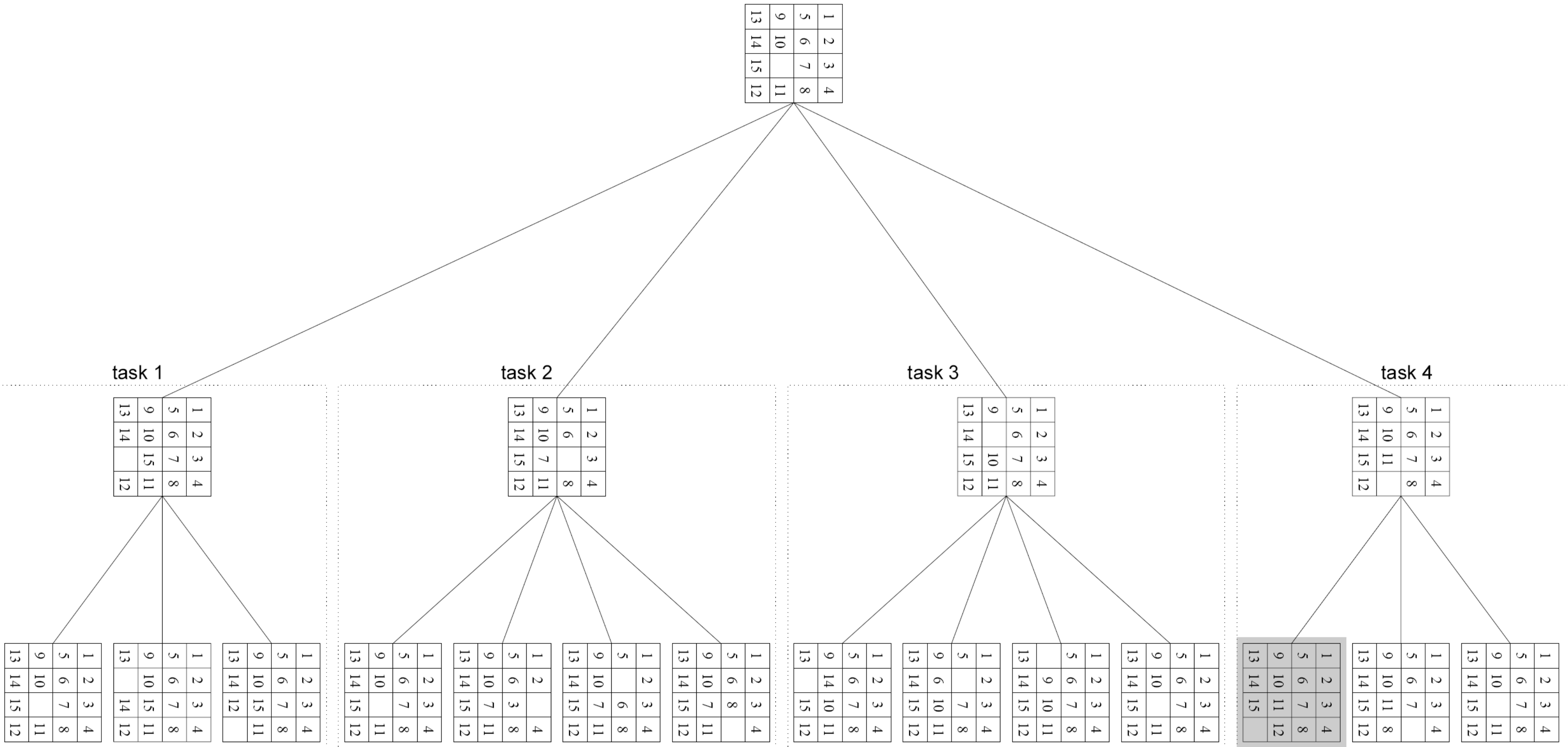
1	2	3	4
5	6	7	8
9	10	11	
13	14	15	12

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

konfiguracja końcowa (rozwiązanie)

- Rozwiązanie problemu polega na przeszukaniu pewnego drzewa (z każdej konfiguracji możemy osiągnąć 4,3 lub dwie inne)

15 - układanka - dekompozycja eksploracyjna



- Dekompozycja eksploracyjna różni się od rekurencyjnej, tym że niezakończone zadania są kończone, gdy tylko zostanie znalezione rozwiązanie.