# Podstawy projektowania algorytmów równoległych

Wojciech Wieczorek

Optymalizacja z użyciem klastrów komputerowych

#### Dekompozycje problemu

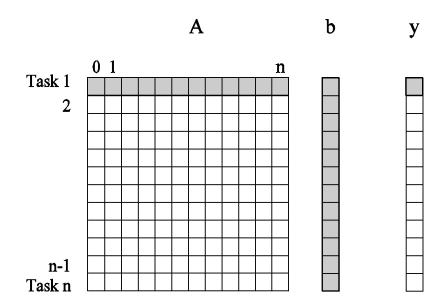
- Dekompozycja funkcjonalna
- Dekompozycja danych
- Dekompozycja rekursywna
- Dekompozycja eksploracyjna
- Dekompozycja spekulatywna

Uwaga: praktyczne obliczenia równoległe są często wieloetapowe. Na każdym etapie mogą być stosowane różne rodzaje dekompozycji. Mamy wówczas do czynienia z dekompozycją hybrydową (mieszaną).

#### Dekompozycja, zadania i graf zależności między zadaniami

- Pierwszym krokiem jest dokonanie dekompozycji problemu obliczeniowego na zadania, które mogą być wykonywane jednocześnie.
- Zadany problem może być zdekomponowany na różne sposoby (warto przeprowadzić analizę rozdrobnienia obliczeń).
- Zadania mogą mieć różne "rozmiary".
- Dekompozycję można zilustrować za pomocą grafu skierowanego, w którym wierzchołki reprezentują zadania, a krawędzie wskazują, że wynik danego zadania jest potrzebny do rozpoczęcia obliczeń w innym zadaniu. Taki graf nazywamy grafem zależności między zadaniami.

#### Przykład: Mnożenie macierzy przez wektor



Wyznaczenie każdego elementu wynikowego wektora **y** nie zależy od innych elementów tego wektora. Można zatem mnożenie to zdekomponować na **n** zadań (ang. tasks).

#### Przykład: przetwarzanie zapytań do baz danych

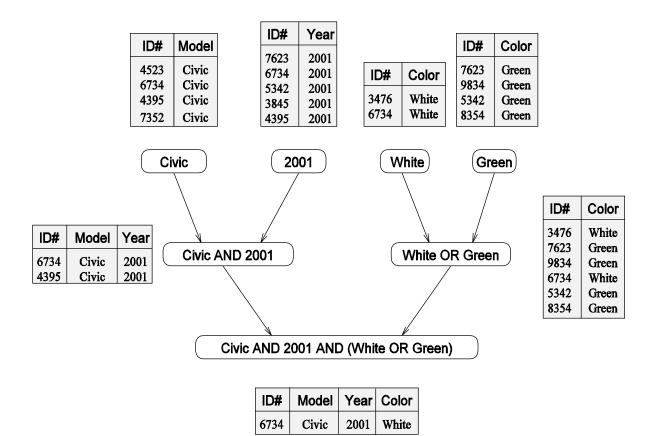
#### Rozważmy kwerendę:

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

#### oraz następującą relację:

ID#	Model	Year	Color	Dealer	Price
4523	3 Civic	2002	Blue	MN	\$18,000
3476	6 Corolla	1999	White	IL	\$15,000
7623	3 Camry	2001	Green	NY	\$21,000
9834	1 Prius	2001	Green	CA	\$18,000
6734	1 Civic	2001	White	OR	\$17,000
5342	2 Altima	2001	Green	FL	\$19,000
3845	5 Maxima	2001	Blue	NY	\$22,000
8354	4 Accord	2000	Green	VT	\$18,000
4395	5 Civic	2001	Red	CA	\$17,000
7352	2 Civic	2002	Red	WA	\$18,000

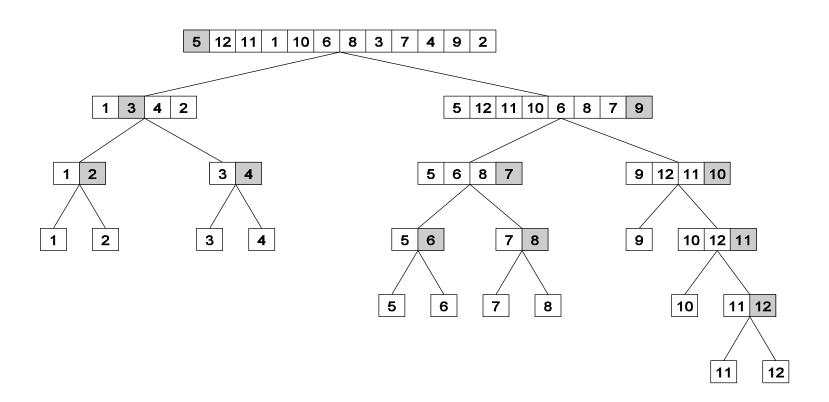
### Jedna z propozycji: dekompozycja rekursywna



#### Dekompozycja rekursywna

- Używana jest zazwyczaj, gdy problem rozwiązywany jest metodą "dziel i zwyciężaj".
- W każdym kroku metody problem dzielony jest na pewną liczbę niezależnych podproblemów.
- Po rozwiązaniu zdań ich wyniki są scalane w rozwiązanie problemu oryginalnego.

#### Quicksort jako przykład dekompozycji rekursywnej



#### Inny problem: znajdowanie elementu minimalnego

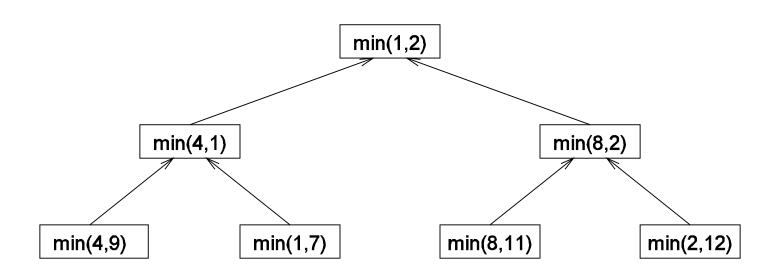
```
    procedure SEKWENCYJNIE_MIN (Am)
    begin
    min = A[0];
    for i:= 1 to n-1 do
    if (A[i] < min) min := A[i];</li>
    endfor;
    return min;
    end SEKWENCYJNIE_MIN
```

#### Dekompozycja rekursywna funkcji MIN

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

#### Dekompozycja rekursywna dla problemu MIN

Poniżej przedstawiono graf zależności między zadaniami dla tablicy A = [4, 9, 1, 7, 8, 11, 2, 12].



#### Uwaga na temat dekompozycji rekursywnej

- Typowe operacje realizowanych za pomocą tej dekompozycji zostały zaimplementowane jako standardowe podprogramy.
- Przykłady:

```
- MPI Reduce
```

- OpenMP: #pragma omp parallel for reduction(+:s)

- CUDA: thrust::reduce

#### Dekompozycja danych

- Najczęściej stosowany sposób wyodrębniania zadań.
- Używa się jej z reguły wtedy, gdy przetwarzane są duże ilości danych.
- Polega ona na podzieleniu danych na pewną liczbę w miarę równych części. Każda z tych części przetwarzana jest przez odrębne zadanie, które wyznacza na ich podstawie pewną ilość danych wyjściowych.

#### Przykład realizacji dekompozycji danych

- **Dane wejściowe**: tablica A[0..999999] liczb rzeczywistych oraz funkcja float f(float x)
- Dane wyjściowe: A[i] = f(A[i]) dla i = 0, 1, ..., 999999.
- OpenMP:

```
#pragma omp parallel shared(A)
{
    #pragma omp for
    for (int i=0; i<999999; i++)
        A[i] = f(A[i]);
}</pre>
```

#### Przykład realizacji dekompozycji danych

CUDA:

```
thrust::transform(A.begin(), A.end(), A.begin(), f());
```

#### MPI:

```
float* a = new float[N/p];
MPI_Scatter(A, N/p, MPI_FLOAT, a, N/p, MPI_FLOAT, 0, MPI_COMM_WORLD);
for (i=0; i<N/p; i++)
   a[i] = f(a[i]);
MPI_Gather(a, N/p, MPI_FLOAT, A, N/p, MPI_FLOAT, 0, MPI_COMM_WORLD);
Delete[] a;</pre>
```

#### Dekompozycja eksploracyjna

- Ten rodzaj dekompozycji stosowany jest dla problemów, w których przeszukiwana jest przestrzeń rozwiązań.
- Przestrzeń dzielona jest na fragmenty eksplorowane równolegle przez poszczególne zadania.
- W momencie znalezienia rozwiązania przez jedno z zadań wysyła ono do pozostałych zadań sygnał o zakończeniu obliczeń.

## Przykład problemu odpowiedniego dla dekompozycji eksploracyjnej

Na rysunku widzimy wybraną pozycję w łamigłówce (a) oraz ruchy prowadzące do pożądanego układu (d).

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

1	2	3	4
5	6	7	8
9	10	$\Diamond$	-11
13	14	15	12

1	2	3	4
5	6	7	8
9	10	11	<b>\</b>
13	14	15	12

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

(a)

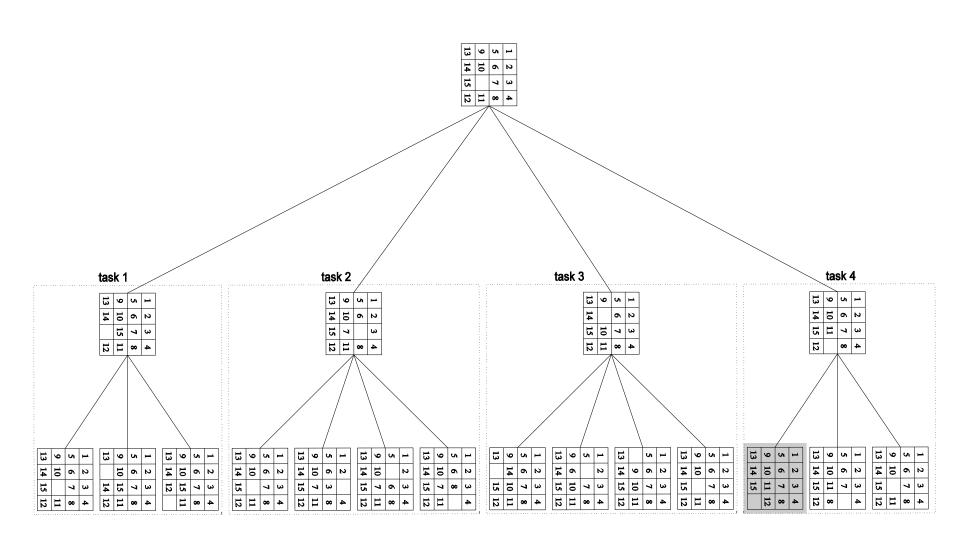
(b)

(c)

(d)

#### Ilustracja dekompozycji eksploracyjnej

W następujący sposób przeszukujemy całą przestrzeń rozwiązań.



#### Dekompozycja spekulatywna

- W niektórych aplikacjach zależności między zadaniami nie są znane a-priori.
- Można wtedy przystąpić do wykonania wielu zadań, mimo iż z góry wiadomo, że wyniki rozwiązania wszystkich zadań z wyjątkiem jednego zostaną odrzucone.

#### Przykład dekompozycji spekulatywnej

```
switch (x = A()) {
  case 0: return B();
  case 1: return C();
A teraz równolegle:
#pragma omp parallel sections
  #pragma omp section
  x = A();
  #pragma omp section
  b = B();
  #pragma omp section
  c = C();
return x == 0 ? b : c;
```

#### Dekompozycja funkcjonalna

- W dekompozycji tej podstawą wyodrębnienia zadań są obliczenia składające się na rozwiązanie problemu.
- Obliczenia te dzielone są na części, zwane funkcjami, realizowane równolegle przez odrębne zadania.
- Zwykle obliczenia poszczególnych funkcji są różne i dotyczą różnych danych.

#### Sito Eratostenesa: kolejne kroki działania

```
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
2 3 4 5 Ø 7 Ø 9 XØ 11 XZ 13 X4 15 XØ 17 XØ 19 ZØ 21 ZZ 23 Z4 25
2 3 4 5 Ø 7 Ø Ø XØ 11 XZ 13 X4 XØ XØ 17 XØ 19 ZØ ZX ZZ 23 Z4 Z5
2 3 4 5 Ø 7 Ø Ø XØ 11 XZ 13 X4 XØ XØ 17 XØ 19 ZØ ZX ZZ 23 Z4 ZØ
```

### Sito Eratostenesa: graf zależności zadań

