

Wykład 2

Podstawowe pojęcia związane z przetwarzaniem równoległym, Podstawowe transformacje pętli, 2 godz.

Plan

1. Transformacja FAN
2. Transformacja PAR
3. Transformacja FAN+PAR
4. Transformacja PIPE
5. Programowanie inkrementacyjne

Podstawowe transformacje pętli

Dla większości kodów, pętle reprezentują najwięcej obliczeń.

Dlatego wykrywanie równoległości powinno odbywać się w pierwszej kolejności w pętlach.

Podstawowe transformacje pętli

Celem tego wykładu jest przedstawienie najbardziej popularnych transformacji pętli programowych pozwalających na ich zrównoleglenie.

Transformacja pętli zmienia kolejność wykonywania iteracji pętli bez dodawania lub usuwania żadnej instrukcji ciała pętli.

Podstawowe transformacje pętli

Transformacja pętli jest legalna jeśli w nowej pętli są honorowane wszystkie zależności: najpierw są wykonywane obliczenia skojarzone z początkiem zależności, dopiero potem są wykonywane obliczenia związane z końcem zależności.

Podstawowe transformacje pętli

Transformacja FAN (Wentylator)

Warunek zastosowania: brak zależności pomiędzy iteracjami pętli.

Sposób wykonywania równoległego:

- 1) każda iteracja może być (ale nie musi) wykonywana niezależnie od pozostałych iteracji;
- 2) żadna synchronizacja obliczeń nie jest wymagana.

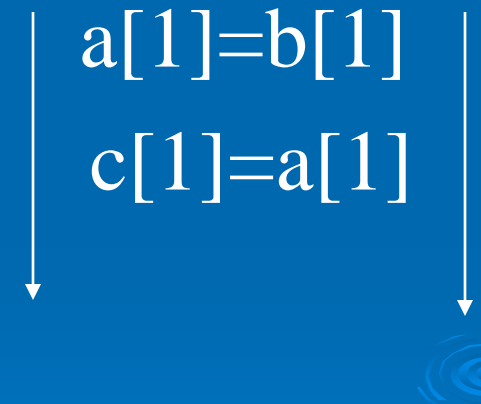
Podstawowe transformacje pętli

Transformacja FAN

Przykład:

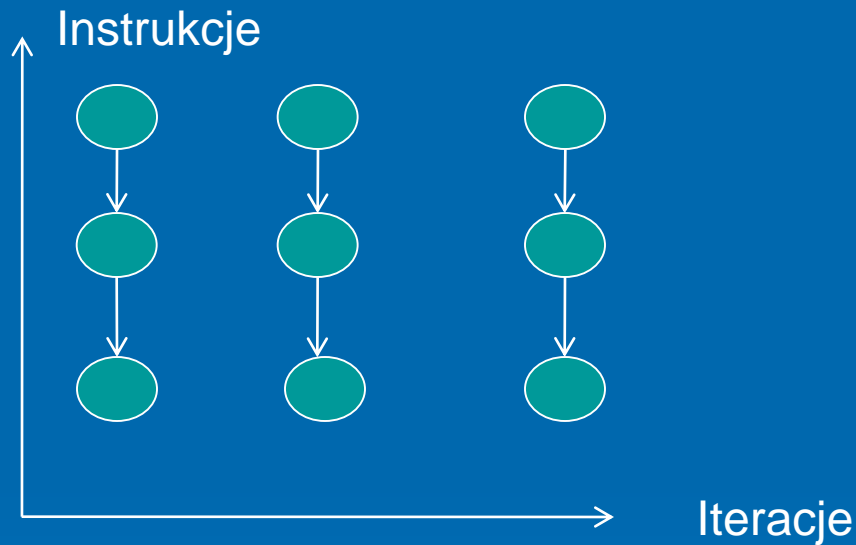
for(i=0; i<=n; i++) { Wykonywanie równoległe

a[i]=b[i];
c[i]=a[i];}

a[0]=b[0]		a[1]=b[1]
c[0]=a[0]		c[1]=a[1]
		

Podstawowe transformacje pętli

Transformacja FAN



Podstawowe transformacje pętli

Transformacja FAN

Właściwa interpretacja zawartości rysunku:

mimo tego, że mogą być zależności pomiędzy instrukcjami w ciele pętli, brak zależności pomiędzy iteracjami pętli umożliwia zastosowanie transformacji FAN.

Podstawowe transformacje pętli

Transformacja FAN

Stopień równoległości jest maksymalną liczbą instrukcji / iteracji / fragmentów kodu, które mogą być wykonywane równolegle.

Stopień równoległości, na który pozwala transformacja FAN, określa się liczbą iteracji pętli.

Podstawowe transformacje pętli

Transformacja FAN

Dla pętli:

```
for(i=1; i<=n; i++) {  
    a[i]=b[i];  
    c[i]=a[i];}
```

Stopień równoległości wynosi n .

Podstawowe transformacje pętli

Transformacja FAN

Dla pętli:

```
for(i=1; i<=n; i++)  
    for(j=1; j<=n; j++) {  
        a[i][j]=b[i][j];  
    }
```

Stopień równoległości wynosi n^2 .

Podstawowe transformacje pętli

Transformacja PAR

Warunki zastosowania:

1. Brak zależności pomiędzy instrukcjami w ciele pętli.
2. Ograniczenie praktyczne: musi być co najmniej dwie instrukcje w ciele pętli.

Podstawowe transformacje pętli

Transformacja PAR

Sposób wykonywania równoległego:

w każdej iteracji, instrukcje ciała pętli mogą być wykonywane niezależnie;

synchronizacja barierowa jest wymagana po wykonaniu każdej iteracji.

Podstawowe transformacje pętli

Transformacja PAR

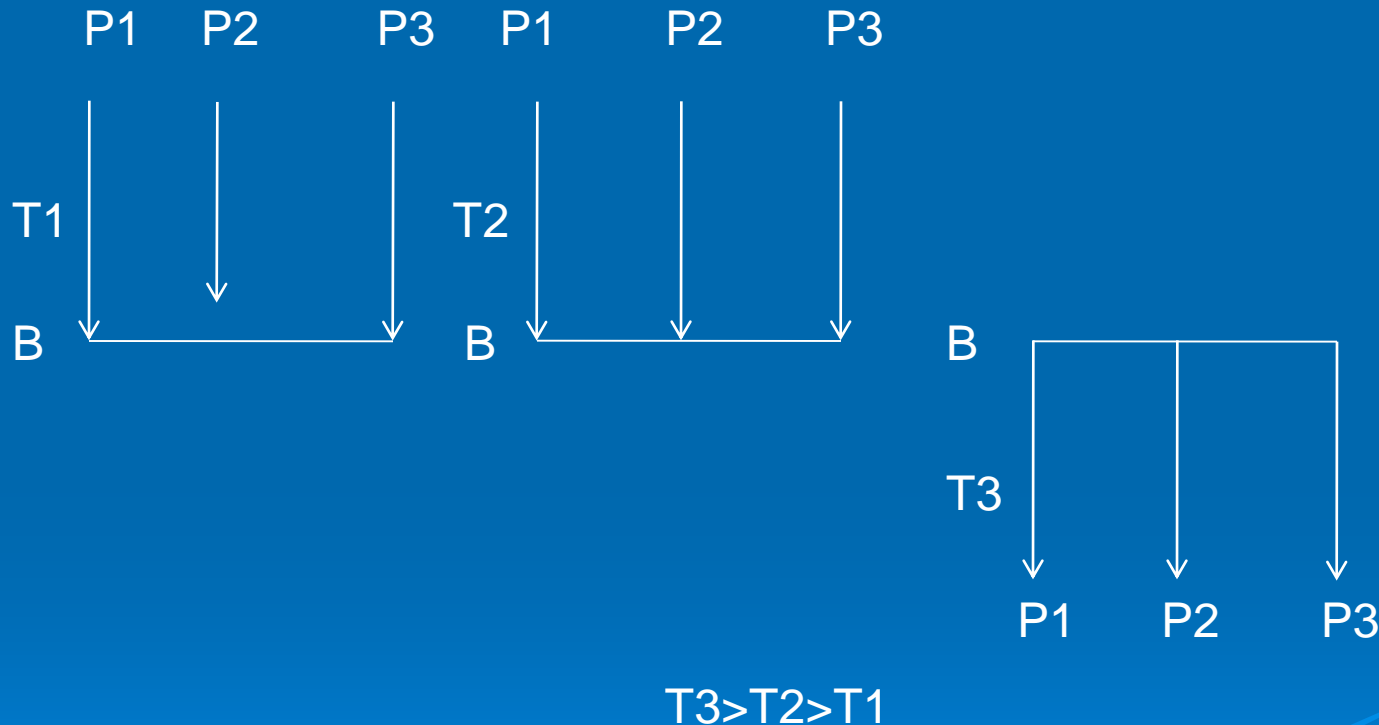
Co to jest synchronizacja barierowa?

To jest punkt w programie, w którym każdy wątek/proces musi czekać na zakończenie wykonania kodu przed tym punktem przez wszystkie pozostałe wątki/procesy.

Następnie wątki/procesy mogą kontynuować wykonywanie kodu znajdującego się po tym punkcie.

Podstawowe transformacje pętli

Transformacja PAR



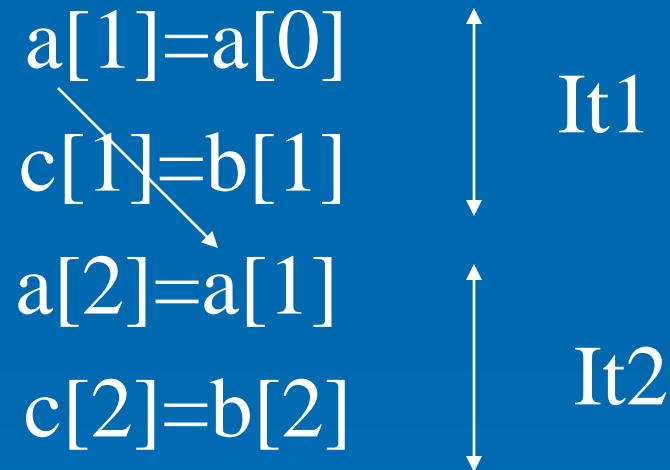
Podstawowe transformacje pętli

Transformacja PAR

➤ Przykład:

```
for(i=1; i<=n; i++) {  
    a[i]=a[i-1];  
    c[i]=b[i];}
```

a[1]=a[0]	↕	It1
c[1]=b[1]		
a[2]=a[1]	↕	It2
c[2]=b[2]		



Podstawowe transformacje pętli

Transformacja PAR

➤ Wykonywanie równoległe:

$\tau_1: a[1]=a[0] \quad c[1]=b[1] \quad IT_1$

Bariera _____

$\tau_2: a[2]=a[1] \quad c[2]=b[2] \quad IT_2$

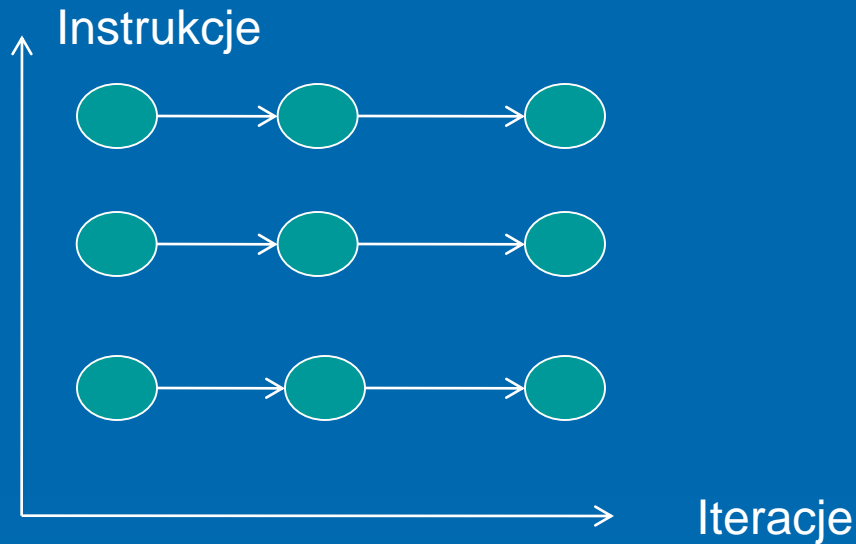
Bariera _____

.....

$T_2 > T_1$

Podstawowe transformacje pętli

Transformacja PAR



Podstawowe transformacje pętli

Transformacja PAR

Właściwa interpretacja zawartości rysunku z poprzedniego slajdu:

mimo tego, że mogą być zależności pomiędzy iteracjami pętli, brak zależności pomiędzy instrukcjami w ciele pętli umożliwia zastosowanie transformacji PAR.

Podstawowe transformacje pętli

Transformacja PAR

- Poziom równoległości, na który pozwala transformacja PAR, określa się liczbą instrukcji ciała pętli.
- Dla pętli niżej:

```
for(i=1; i<=n; i++) {  
    a[i]=a[i-1];  
    c[i]=b[i];}
```

stopień równoległości wynosi 2.

Podstawowe transformacje pętli

Transformacja PAR

Kod pętli po transformacji PAR:

Dyrektywa określająca
możliwość
wykonywania kodu w
sposób równoległy

```
for(i=1; i<=n; i++){  
    a[i]=a[i-1];  
    c[i]=b[i];}
```



```
for(i=1; i<=n; i++) {  
    parallel {  
        a[i]=a[i-1];  
        c[i]=b[i];}  
    barrier}
```

Podstawowe transformacje pętli

Transformacja FAN+PAR

Warunek zastosowania: brak zależności pomiędzy iteracjami pętli i brak zależności pomiędzy instrukcjami w ciele pętli.

Podstawowe transformacje pętli

Transformacja FAN+PAR

Sposób wykonywania równoległego:

każda instrukcja ciała pętli każdej iteracji
może być wykonywana niezależnie;
nie jest wymagana żadna synchronizacja.

Podstawowe transformacje pętli

Transformacja FAN+PAR

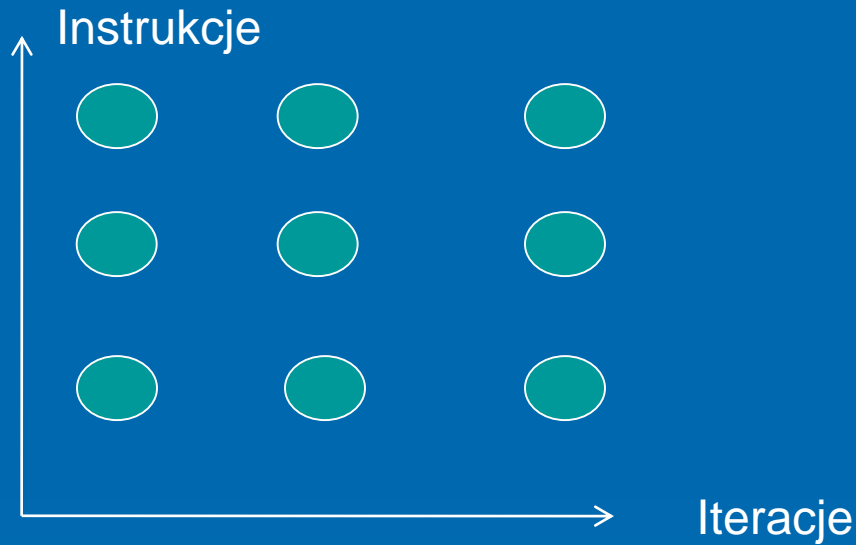
```
for(i=0; i<=n; i++) {  
    a[i]=b[i];  
    c[i]=d[i];}
```

Wykonywanie równoległe:

P1	P2	P3	P4
a[0]=b[0]	c[0]=d[0]	a[1]=b[1]	c[1]=d[1]

Podstawowe transformacje pętli

Transformacja FAN+PAR



Podstawowe transformacje pętli

Transformacja FAN+PAR

Stopień równoległości, który umożliwia transformacja FAN + PAR, jest iloczynem liczby iteracji pętli i liczby instrukcji w ciele pętli.

Podstawowe transformacje pętli

Transformacja FAN+PAR

Stopień równoległości dla pętli:


```
for(i=1; i<=n; i++) {  
    a[i]=b[i];  
    c[i]=d[i];}
```

wynosi $2n$.

Podstawowe transformacje pętli

Transformacja FAN+PAR

➤ Kod pętli po transformacji FAN+PAR:

<pre>for(i=1; i<=n; i++){ a[i]=b[i]; c[i]=d[i];}</pre>		<pre>parfor(i=1; i<=n; i++){ parallel{ a[i]=b[i]; c[i]=d[i]; } }</pre>
---	---	---

Podstawowe transformacje pętli

Transformacja PIPE (Potok)

Rozważmy następującą pętlę:

```
for (i=1; i<=n; i++) {
```

```
    a[i]=a[i-1];
```

```
    b[i]=a[i];
```

```
}
```

a[1]=a[0]

b[1]=a[1]

a[2]=a[1]

b[2]=a[2]

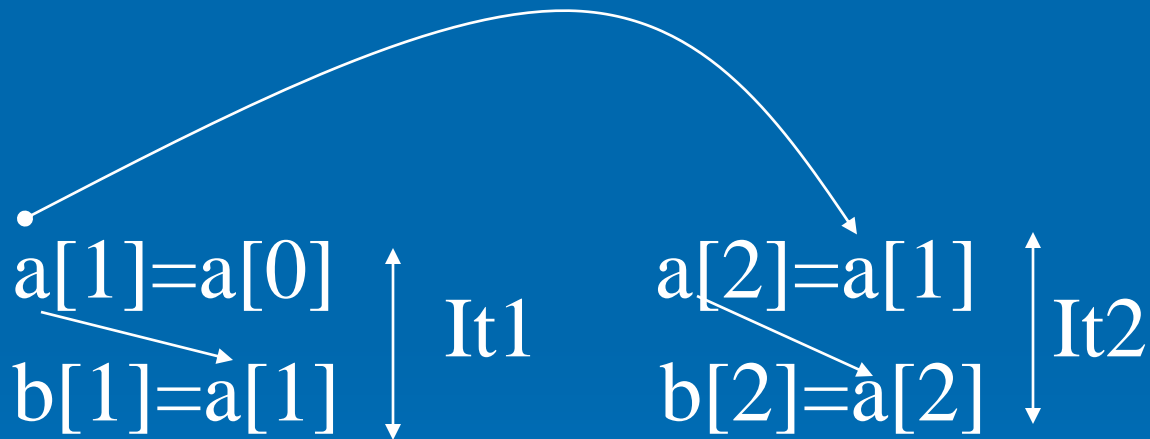
↑
It1
↓

↑
It2
↓

Podstawowe transformacje pętli

Transformacja PIPE

Rozważmy następujący sposób obliczeń:

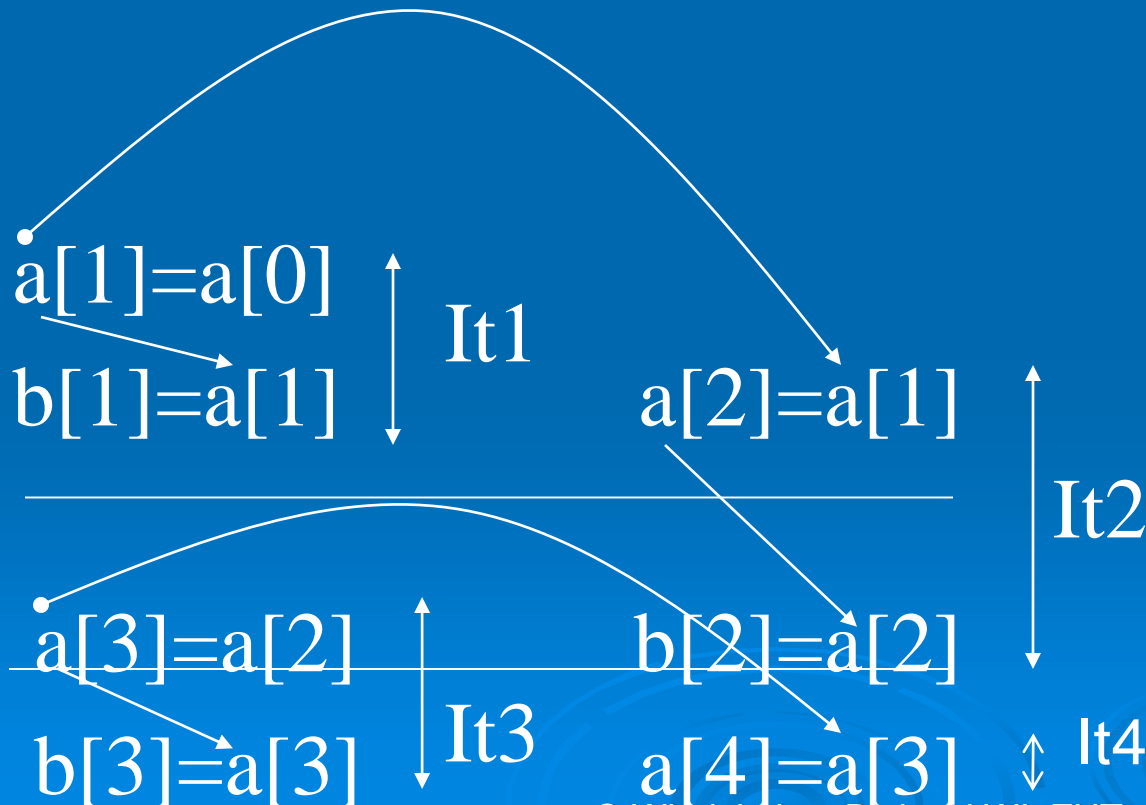


Takie wykonywanie nie jest poprawne.

Podstawowe transformacje pętli

Transformacja PIPE

Rozważmy kolejny sposób obliczeń:



Podstawowe transformacje pętli

Transformacja PIPE

Sposób wykonywania równoległego dla k wątków/procesów:

1. Przydziel iterację 1 do wątku/procesu 1

Przydziel iterację 2 do wątku/procesu 2

.....

Przydziel iterację k do wątku/procesu k

Podstawowe transformacje pętli

Transformacja PIPE

Sposób wykonywania równoległego dla k wątków/procesów:

Przydziel iterację $k+1$ do wątku/procesu 1

Przydziel iterację $k+2$ do wątku/procesu 2

.....

Przydziel iterację $2k$ do wątku/procesu k

.....

Podstawowe transformacje pętli

Transformacja PIPE

2. Określ opóźnienie pomiędzy wykonywaniem pary kolejnych iteracji tak aby wszystkie zależności były honorowane, tj. dla każdej zależności początek zależności ma być wykonywany wcześniej niż koniec tej zależności, czyli zależne instrukcje nie mogą być wykonywane jednocześnie.

Podstawowe transformacje pętli

Transformacja PIPE

Opóźnienie może przyjąć jedną z następujących wartości:

$0, 1, 2, \dots, M-1,$

gdzie M jest to liczba instrukcji w ciele pętli.

Ze wszystkich możliwych opóźnień należy wybrać najmniejsze.

Podstawowe transformacje pętli

Transformacja PIPE

3. Wstaw synchronizację barierową po wykonaniu każdej instrukcji.

Podstawowe transformacje pętli

Transformacja PIPE

- Stopień równoległości, PD (*parallelism degree*), na który pozwala transformacja PIPE, jest uzależniony od:
 - liczby wątków/procesów k ,
 - liczby instrukcji w ciele pętli M ,
 - opóźnień D (*delaying*),
 - liczby iteracji N .

Podstawowe transformacje pętli

Transformacja PIPE

Wzory do określenia stopnia równoległości:

$PD = k$, jeśli $N \geq k$ oraz $D=0$

$PD = M - D + 1$, jeśli $M - D > 0$

$PD = 1$, jeśli $M - D \leq 0$.

Podstawowe transformacje pętli

Transformacja PIPE

Tabela niżej pokazuje wartości stopnia równoległości w zależności od k , D i M

$D \backslash M$	1	2	3	4
0	k	k	k	k
1	1	2	3	4
2	1	1	2	3
3	1	1	1	2

Podstawowe transformacje pętli

Transformacja PIPE

Czy transformacja PIPE jest równoważna z transformacją FAN dla opóźnienia $D=0$?

Podstawowe transformacje pętli

Transformacja PIPE

Nie, mamy następujące różnice:

1. Po transformacji PIPE iteracje są przypisane do określonych z góry wątków/procesów, natomiast po transformacji FAN iteracje mogą być przypisane do dowolnego wątku/procesu.
2. PIPE wymaga synchronizacji barierowej po każdej instrukcji ciała pętli.

Podstawowe transformacje pętli

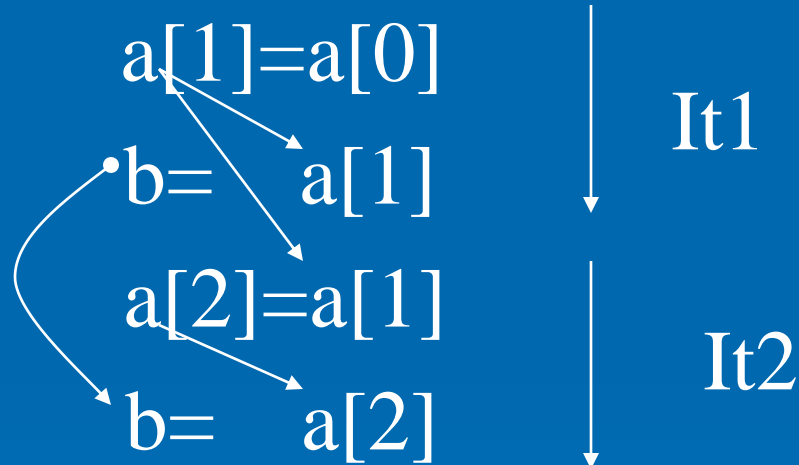
Przykład

```
for (i=1; i<=n; i++) {
```

```
    a[i]=a[i-1];
```

```
    b=a[i];
```

```
}
```



FAN - Nie

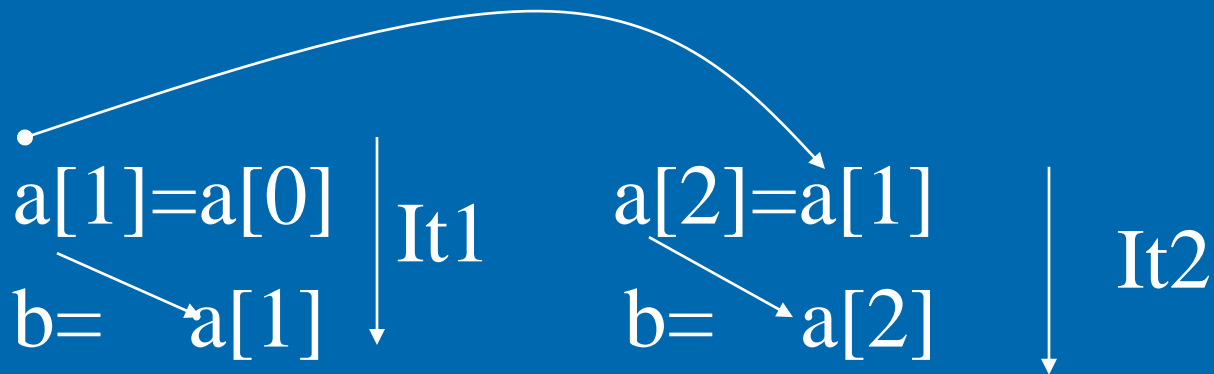
PAR - Nie

PIPE - ?

Podstawowe transformacje pętli

Przykład

$D = 0$

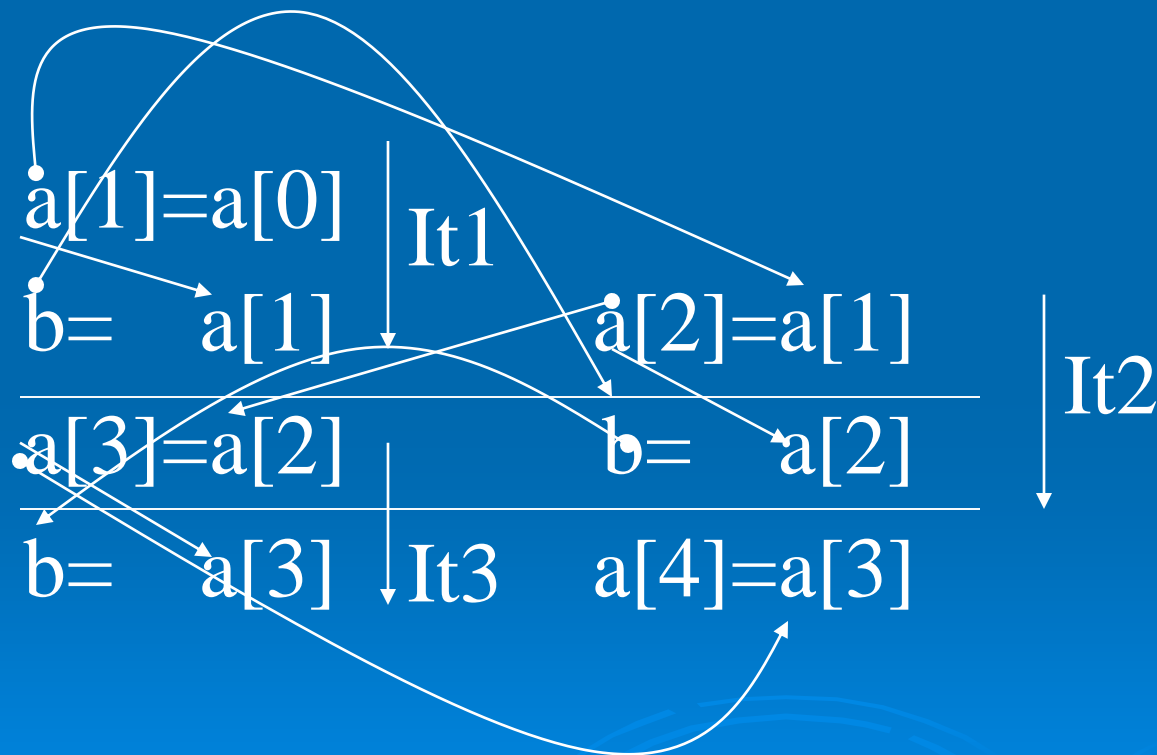


Wartość $D=0$ nie jest właściwa

Podstawowe transformacje pętli

Przykład

$D = 1$



Wartość $D=1$ jest właściwa

Podstawowe transformacje pętli

Kod po transformacji PIPE

Dla pętli:

```
for (i=1; i<=n; i++) {  
    a[i]=a[i-1];  
    b=a[i];  
}
```

już wiemy, że $D=1$, $M=2$, czyli maksymalna liczba wątków/procesów dla transformacji PIPE tej pętli wynosi 2.

Podstawowe transformacje pętli

Kod po transformacji PIPE jest następujący:

```
parfor(k=0; k<2; k++)    //przebiera wątki
  for (i=k; i<=n; i=i+2) { //przebiera iteracje
    if (k > 0) wait (k-1); // czeka na sygnał
    a[i]=a[i-1];
    b=a[i];
    if (k < 1 ) signal (k+1); //wysyła sygnał
  }
```


Podstawowe transformacje pętli

gdzie funkcja `wait(k-1)` razem z funkcją `signal(k+1)` implementuje opóźnienie przez czekanie na sygnał od wątku/procesu $k-1$ i może być zaimplementowana w zależności od wybranego API lub języka programowania równoległego na różne sposoby za pomocą pętli `while`, semaforów, zamków i t.d.

Podstawowe transformacje pętli

- Funkcja **signal(k+1)** ustawia odpowiednią wartość zmiennej, semafora, zamka, z której korzysta funkcja **wait(k-1)** wątku/procesu k+1.

Programowanie inkrementacyjne

Polega na tym, że mając na wejściu program sekwencyjny uzupełniamy go za pomocą dyrektyw, funkcji oraz zmiennych środowiskowych.

Program rozbudowany (równoległy) wciąż może być kompilowany jako sekwencyjny.

Składowe OpenMP

Dyrektwy

Parallel

Podziału pracy

Synchronizacji

Klauzule zakresu
zmiennych:

- private
- firstprivate
- last private
- shared
- reduction

Funkcje czasu wykonania

Liczba wątków

Identyfikator wątku

Wątki dynamiczne

Równoległość zagnieżdżona

Czasomierze

API dla zamków

Zmienne środowiskowe

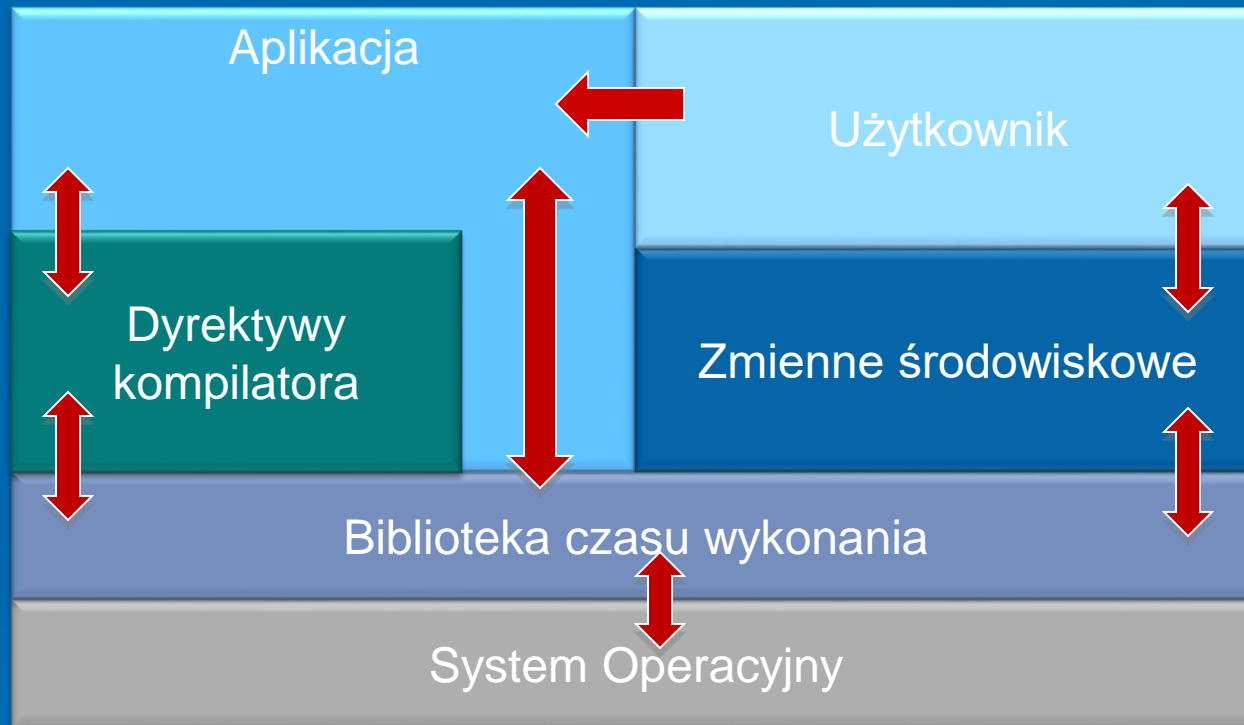
Liczba wątków

Typ szeregowania

Wątki dynamiczne

Równoległość
zagnieżdżona

Architektura OpenMP



Dziękuję za uwagę