Tworzenie programów równoległych

Tworzenie programów równoległych

- W procesie tworzenia programów równoległych istnieją dwa kroki o zasadniczym znaczeniu:
  - **wykrycie dostępnej współbieżności w trakcie realizacji programu**
  - **określenie koniecznej synchronizacji lub wymiany komunikatów pomiędzy procesami lub wątkami realizującymi program**
- Pierwszy z tych kroków często ma charakter bardziej twórczy, drugi bardziej techniczny (zakłada znajomość modelu programowania)
- Jednym z najważniejszych wymagań stawianych programom równoległym jest przenośność – **możliwość uruchomienia na maszynie o dowolnej liczbie procesorów/rdzeni**
  - **liczba wątków/procesów staje się parametrem programu**

Metodologia programowania równoległego

- Przydatnym sposobem ujęcia metodologii programowania równoległego jest wyróżnienie pięciu podstawowych zadań, które muszą zostać zrealizowane przy tworzeniu programu równoległego:
  - podział (dekompozycja) zadania obliczeniowego na podzadania
  - odwzorowanie zadań na procesy i wątki oraz dalej na elementy przetwarzania: węzły, multi-procesory, procesory (rdzenie)
  - podział (dystrybucja) danych pomiędzy elementy pamięci związane z elementami przetwarzania (uwzględniając hierarchię pamięci)
  - określenie koniecznej wymiany danych między procesami (wątkami) oraz odwzorowanie jej na sieć połączeń między procesorami (rdzeniami)
  - określenie koniecznej synchronizacji między zadaniami (=wątkami (procesami))

Metodologia programowania równoległego

- Przykłady podziałów zadania na podzadania:
  - Podział ze względu na funkcje *(**functional decomposition**):*
    - standardowe złożone aplikacje (np. z obsługą plików, korektą poprawności w trakcie pracy itp.)
    - złożony problem optymalizacji
  - Podział struktury danych (***data decomposition***)
    - sortowanie tablic
    - rozwiązywanie układów równań liniowych
  - Podział w dziedzinie problemu *(**domain decomposition***)
    - symulacje zjawisk fizycznych w przestrzeni (wykorzystanie podziału geometrycznego – ***geometric decomposition***)
    - rozwiązywanie układów równań liniowych
  - Podział złożony, mający cechy i podziału funkcjonalnego i danych:
    - przetwarzanie potokowe (np. sekwencji obrazów)

Przykłady

- Praktyczne przykłady dekompozycji (nomenklatura standardu POSIX jako przykładowej realizacji):
  - funkcjonalna:

```
pthread_create(&w1, NULL, funkcja_1, &arg_f1);
pthread_create(&w2, NULL, funkcja_2, &arg_f2);
.... // itd.
```

  - prosta dekompozycja pętli (SPMD):

```
for(i=0; i<N; i++){
   pthread_create(&w[i], NULL, funkcja, &arg_f[i]); // często przesyłany jako argument
}                                                   // ID wątku od 0 do p-1
```

  - bardziej złożone dekompozycje pętli: cykliczne, blokowe, mieszane itp.
    - liczba wątków niezależna od liczby iteracji
  - dekompozycja zadań (SPMD) – tyle wątków ile zadań:

```
for(i=0; i<p; i++){
   pthread_create(&w[i], NULL, funkcja, &definicja_zadania[i]);
}
```

  - klasyfikacja jest nieostra, rodzaje dekompozycji zachodzą na siebie

Prosty przykład

```
#include<pthread.h>
#define N 100 // liczba iteracji (elementów przetwarzanej tablicy): N – jako parametr
#define LICZBA_W_MAX 44 // maksymalna liczba wątków

void *suma_w( void *arg_wsk)
int p; pthread_mutex_t muteks; int suma=0; pthread_t watki[LICZBA_W_MAX];

int main( int argc, char *argv[] ){
      int i; int indeksy[LICZBA_W_MAX];
      printf("Podaj liczbę wątków: "); scanf("%d",&p);
      for(i=0;i<p;i++) indeksy[i]=i;
      pthread_mutex_init( &muteks, NULL);
      for(i=0; i<p; i++ ) pthread_create( &watki[i], NULL, suma_w, (void *) &indeksy[i] );
      for(i=0; i<p; i++ ) pthread_join( watki[i], NULL );
      printf(„suma = %d\n”,suma);
}
```

Prosty przykład

```
void *suma_w( void *arg_wsk){

    int moj_id = *( (int *) arg_wsk ); // od 0 do p-1

    int el_na_watek =  ceil( (float) N / p );
    int moj_start = el_na_watek*moj_id ; // lub moj_start = moj_id
    int moj_koniec = el_na_watek*(moj_id+1) ; // lub moj_koniec = N
    if(moj_koniec > N) moj_koniec = N;
    int moj_skok = 1; // lub moj_skok = p

    for(int i=moj_start; i<moj_koniec; i+=moj_skok)
                moja_suma += i; // lub moja_suma += A[i]; lub np. A[i] = B[i];

    pthread_mutex_lock( &muteks );
    suma += moja_suma;
    pthread_mutex_unlock( &muteks );
    pthread_exit( (void *)0);
}
```

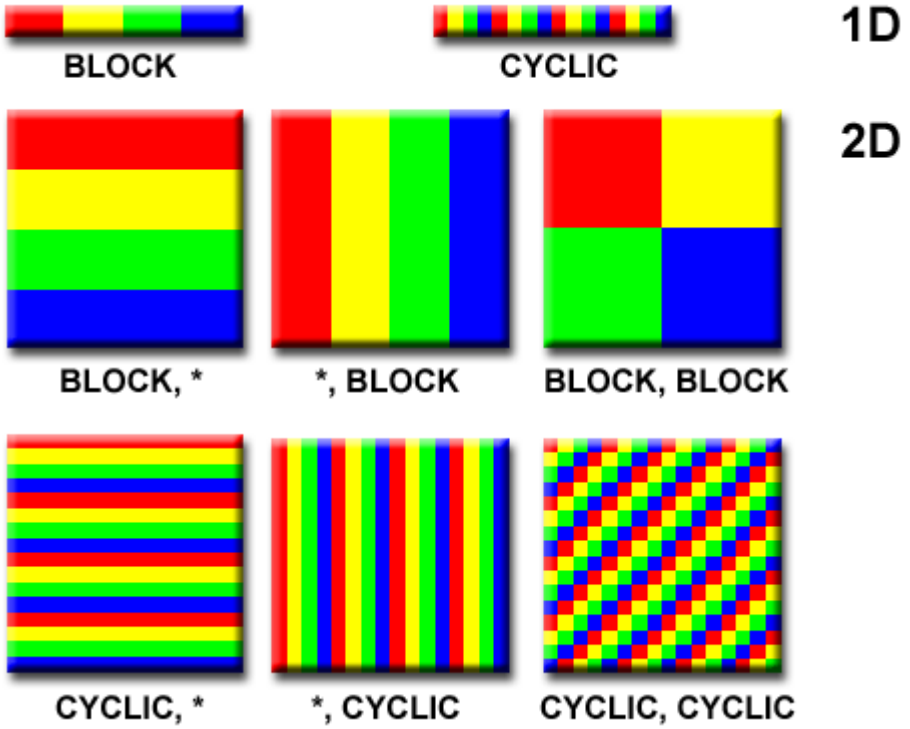
Równoważenie obciążenia

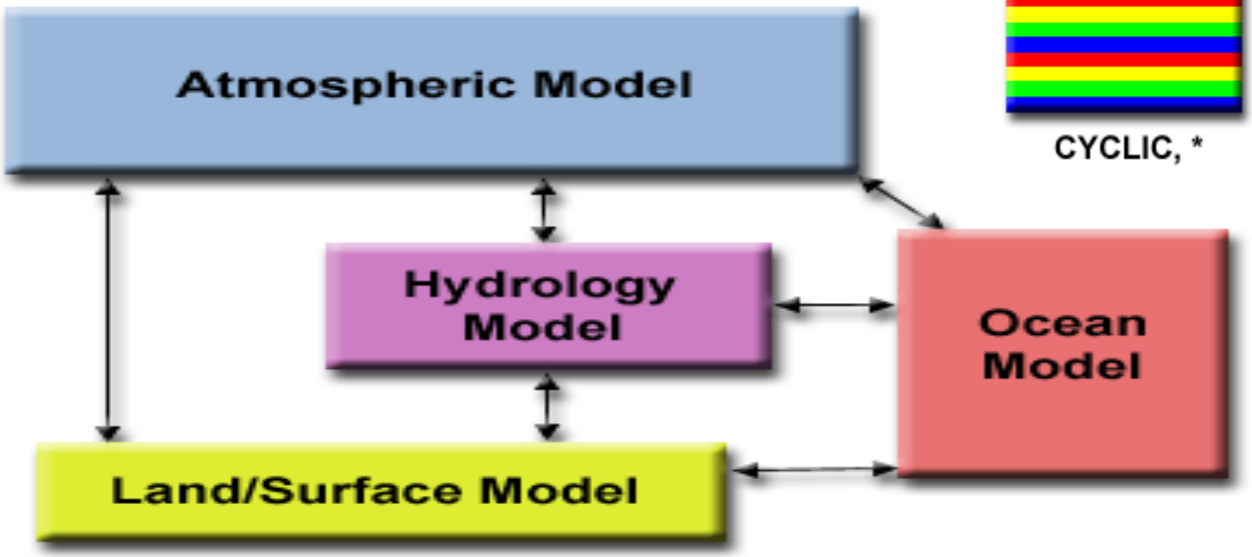
Przypadki typowe	N = 1000100, p = 16	N = 1000100, p = 16
dekompozycja:	blokowa	cykliczna
N/p , ceil (N/p)	62506 , 62507	62506 , 62507
liczba iteracji (elementów) – pierwszy wątek	62507	62507
liczba iteracji (elementów) – ostatni wątek	62495	62506
liczba wątków o mniejszej liczbie iteracji	1	12 (zawsze <p)
maksymalna różnica liczby iteracji na wątek	12 (zawsze <p)	1
maksymalna liczba iteracji na wątek	**62507**	**62507**

Przypadki nietypowe	N = 101, p = 32	N = 101, p = 32
dekompozycja:	blokowa	cykliczna
N/p , ceil (N/p)	3 , 4	3 , 4
liczba iteracji (elementów) – pierwszy wątek	4	4
liczba iteracji (elementów) – ostatni wątek	0	3
liczba wątków o mniejszej liczbie iteracji	7 (6 bezczynnych)	27 (zawsze <p)
maksymalna różnica liczby iteracji na wątek	4 (zawsze <p)	1
maksymalna liczba iteracji na wątek	**4**	**4**

Przykłady

- Algorytmy operujące na tablicach dwuwymiarowych
  - alternatywy dekompozycji danych
  - powiązanie dekompozycji ze sposobem przechowywania macierzy
- Modelowanie środowiska naturalnego





Metodologia programowania równoległego

- „Wzorce” programowania równoległego
  - Zarządca-wykonawcy (***manager-worker, master-slave***)
  - Dziel i rządź (***divide and conquer***) i inne wersje dynamicznego, rekursywnego zarządzania obliczeniami (***recursive splitting*** np. ***fork-join***)
  - Zrównoleglenie pętli (***loop parallelism***)
  - Przetwarzanie potokowe (***pipelining***)
  - Agenci (***software agents***), aktorzy (***actors***), przetwarzanie sterowane zdarzeniami (***discrete event***)
  - Map-Reduce (mikro-wzorzec redukcji)
  - i wiele innych (np. mikro-wzorzec *busy wait*)
- Wzorce są ogólnymi wskazówkami rozwiązania problemu dekompozycji zadania na równoległe pod-zadania, konkretne programy mogą realizować wiele, często odpowiednio zmodyfikowanych, wzorców

Metodologia programowania równoległego

Pattern	OpenMP	OpenCL	MPI
Task Parallelism			
Recursive splitting	with 3.0		
Pipeline			
Geometric Decomposition			
Discrete Event			
SPMD			
SIMD			
Fork/Join			with MPI 2.0
Actors			
BSP			
Loop Parallelism			
Master/worker			

Good mapping
Weak mapping
Not suitable

Prosty przykład sumowania

- Prosty przykład sumowania łączy w sobie kilka wzorców, modeli, sposobów programowania:
  - można go utworzyć na podstawie podziału w dziedzinie problemu lub podziału danych
  - realizuje statyczną równoległość zadań
  - realizuje wzorzec zarządca-wykonawcy
  - wykonanie odbywa się w modelu SPMD
  - jest w rzeczywistości sposobem na zrównoleglenie pętli w środowiskach wielowątkowych

Metodologia programowania równoległego

- Kolejny możliwy podział związany z konkretnymi mechanizmami programowymi:
  - równoległość zadań, ***task parallelism*** – każdy wątek/proces otrzymuje do wykonania pewne zadanie, najczęściej funkcję w programie
    - wątki mogą realizować tę samą funkcję, ale na innych danych lub różne funkcje
    - przydział zadań może być statyczny lub dynamiczny,
    - wykonanie jest najczęściej **asynchroniczne**
  - równoległość wykonania pętli, ***loop parallelism*** – każdy wątek/proces otrzymuje pewną liczbę iteracji pętli do wykonania
    - każdy proces/wątek posiada własny indeks iteracji, który przyjmuje wartości z określonego podzbioru pełnego zbioru wartości indeksów dla pętli
    - wykonanie jest najczęściej **synchroniczne**
- Przykład obliczania całki – możliwe podejścia

Metodologia programowania równoległego

- Ważnym w dziedzinie przetwarzania równoległego jest pojęcie równoległości danych (***data parallelism***)
  - w sensie węższym oznacza model programowania, w którym programista określa jawnie przydział danych procesom/wątkom oraz operację do wykonania na całości danych, natomiast konkretna realizacja obliczeń przez procesy/wątki, w tym konieczna synchronizacja i komunikacja, jest przeprowadzana przez środowisko wykonania, bez jawnego udziału programisty
    - realizacją tego modelu są języki równoległości danych takie jak np. High Performance Fortran (HPF)
  - w sensie szerszym oznacza model w którym podstawą zrównoleglenia jest podział (dekompozycja) danych, odpowiedni przydział danych wątkom/procesom oraz wykonanie zgodnie z zasadą właściciel przeprowadza obliczenia (***owner computes***)

Metodologia programowania równoległego

- W przypadku modelu wykonania równoległego (w ramach konkretnych środowisk programowania równoległego) można rozważać model oparty na dekompozycji danych (***data decomposition***), jako alternatywę dla modelu opartego na równoległości sterowania (***control decomposition***)
  - w modelu **dekompozycji danych** często **wątek/procesor na podstawie swojego indeksu ustala dane, na których operuje,** przy czym wszystkie wątki realizują ten sam kod
  - w modelu **dekompozycji sterowania** często **wątek na podstawie swojego indeksu ustala, którą ścieżkę w programie lub którą iterację w pętli powinien realizować**
  - poza czystymi algorytmami dekompozycji danych i sterowania, występuje wiele przypadków pośrednich (różne dane i częściowo różne ścieżki wykonania dla różnych wątków)

Metodologia programowania równoległego

- Wygodnym sposobem uporządkowania procesu tworzenia programów równoległych jest ujęcie go w ramy specyficznej metodologii
- Jedną z takich metodologii jest PCAM (Foster 1985)
- Kolejne litery oznaczają kroki przy tworzeniu programu:
  - P – *partition*, podział zadania na podzadania
  - C – *communicate,* określenie niezbędnej komunikacji, podział danych na wspólne i prywatne, określenie sposobu korzystania z danych, synchronizacja operacji na danych
  - A – *agglomerate,* analiza wariantów podziału
  - M – *map,* uwzględnienie ostatecznej implementacji, „odwzorowania" na architekturę sprzętu
- Pierwsze dwa kroki zmierzają do stworzenia poprawnego programu równoległego, kolejne dwa do jego optymalizacji