PARADYGMATY I JĘZYKI PROGRAMOWANIA

Programowanie współbieżne ... (w14)

Treść

- Algorytmy równoległe.
  - Metoda sum prefiksowych
  - Mnożenie tablic
- OpenMP
  - Podstawy
- CUDA
  - Informacja

Sumy prefiksowe

- Przykłady z materiałów:
  - Aoyama, Nakano:
    RS/6000 SP: Practical MPI Programming, 1999.
    http://www.redbooks.ibm.com/abstracts/sg245380.html
  - Z. Czech: Wprowadzenie do do obliczeń równoległych. PWN, Warszawa, 2013.

do – metoda sum prefiksowych

(Patrz: Aoyama, Nakano; Czech)


```
Program main
Parameter (n=...)
real a(0:n), b(n)
...
do i=1, n
   a(i) = a(i-1) + b(i)
end do
...
```

Zależności:

a(0) ⇒ a(1) ⇒ a(2) ⇒... ⇒ a(n)

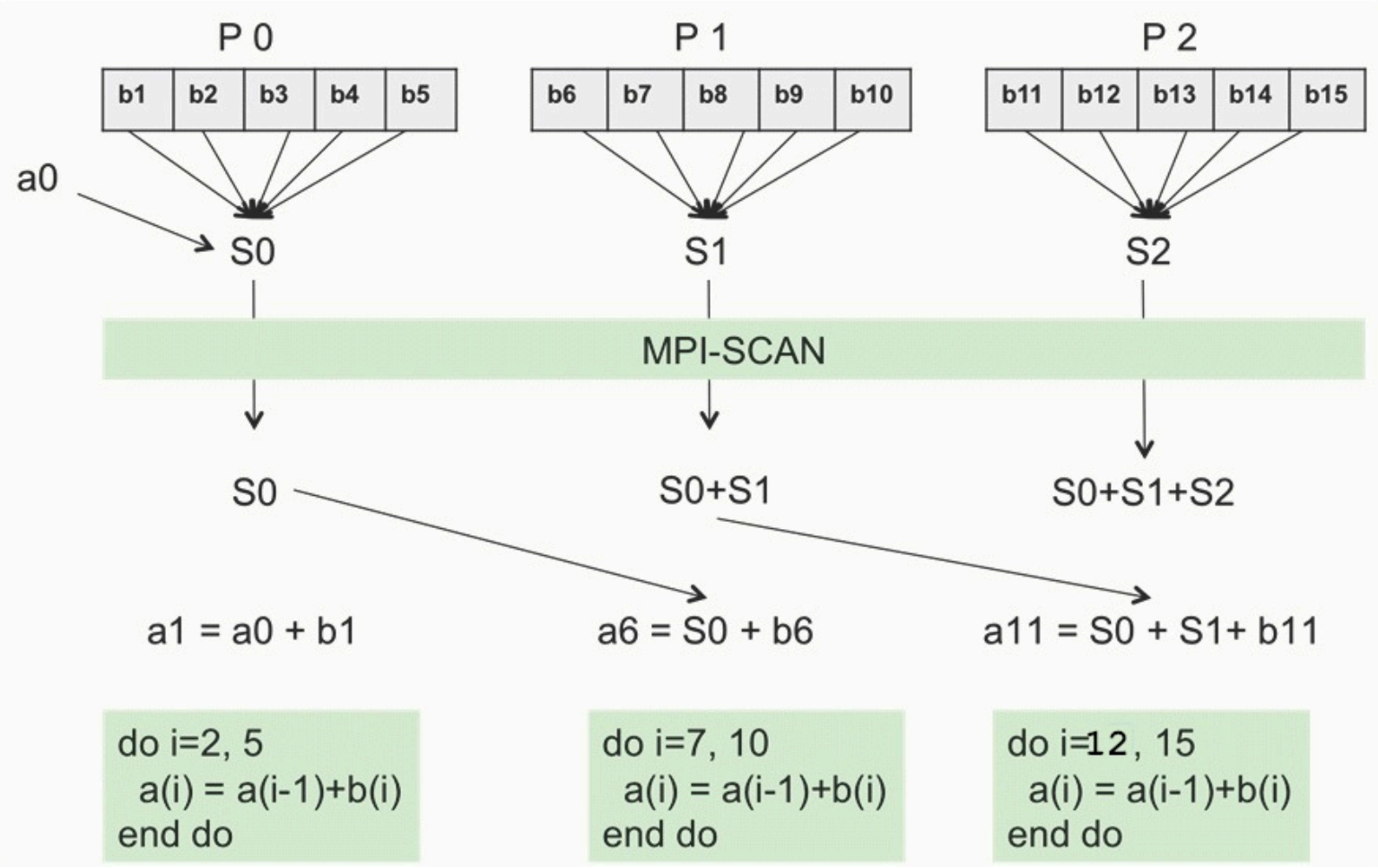
Jawnie:

```
a(1) = a(0) + b(1)
a(2) = a(0) + b(1) + b(2)
a(3) = a(0) + b(1) + b(2) + b(3)
...
a(n) = a(0) + b(1) + ... +  b(n)
```

Ogólniej:

```
a(i)=a(i-1) op b(i)
```

do – metoda sum prefiksowych



do – metoda sum prefiksowych


```
PROGRAM main
INCLUDE 'mpif.h'
PARAMETER (n = ...)
REAL a(0:n), b(n)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
CALL range(1, n, nprocs, myrank, ista, iend)
...
sum = 0.0
DO i = ista, iend
   sum = sum + b(i)
ENDDO
```

do – metoda sum prefiksowych


```
IF (myrank == 0) THEN
   sum = sum + a(0)
ENDIF
CALL MPI_SCAN(sum, ssum, 1, MPI_REAL, MPI_SUM, &
                            MPI_COMM_WORLD, ierr)
a(ista) = b(ista) + ssum - sum
IF (myrank == 0) THEN
   a(ista) = a(ista) + a(0)
END IF
DO i = ista+1, iend
   a(i) = a(i-1) + b(i)
END DO
```

Operacje macierzowe

- Mnożenie: *A x B = C*; np. 4-procesy. Macierze -> podmacierze

$a_{1,1}$	$a_{1,2}$		$b_{1,1}$	$b_{1,2}$		$c_{1,1}$	$c_{1,2}$
$a_{2,1}$	$a_{2,2}$	x	$b_{2,1}$	$b_{2,2}$	=	$c_{2,1}$	$c_{2,2}$

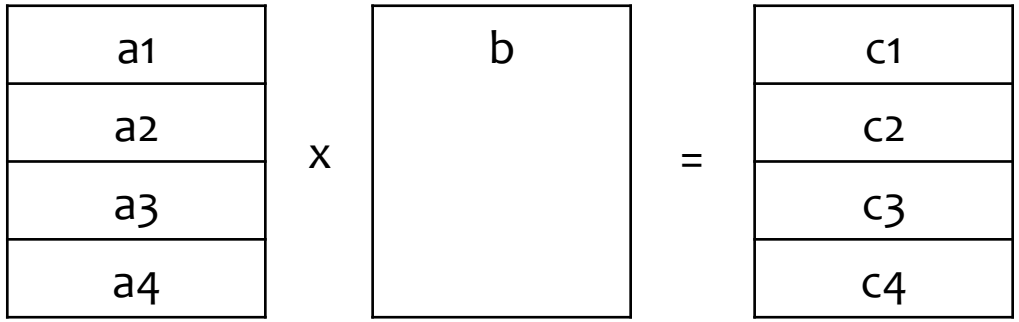
- Zadanie 1: $(c_{1,1}) = (a_{1,1}) \times (b_{1,1}) + (a_{1,2}) \times (b_{2,1})$
- Zadanie 2: $(c_{1,2}) = (a_{1,1}) \times (b_{1,2}) + (a_{1,2}) \times (b_{2,2})$
- Zadanie 3: $(c_{2,1}) = (a_{2,1}) \times (b_{1,1}) + (a_{2,2}) \times (b_{2,1})$
- Zadanie 4: $(c_{2,2}) = (a_{2,1}) \times (b_{1,2}) + (a_{2,2}) \times (b_{2,2})$

Tutaj $(a_{i,j})$ oznacza podmacierz, w szczególności element macierzy A

Stopień współbieżności 4.

Operacje macierzowe

- Mnożenie: *A x B = C*; np. 4-procesy ale inaczej...



- stopień współbieżności 4 (słabo!)

Operacje macierzowe

- LEPIEJ! Rozdrobnienie operacji...

$a_{1,1}$	$a_{1,2}$
$a_{2,1}$	$a_{2,2}$

x

$b_{1,1}$	$b_{1,2}$
$b_{2,1}$	$b_{2,2}$

=

$c_{1,1}$	$c_{1,2}$
$c_{2,1}$	$c_{2,2}$

- Zad1. d1 = a11 b11
- Zad2. d2 = a12 b21
- Zad3. d3 = a11 b12
- Zad4. d4 = a12 b22
- Zad5. d5 = a21 b11
- Zad6. d6 = a21 b21

Zad7. d7 = a21 b12
Zad8. d8 = a22 b22
Zad9. c11 = d1 + d2
Zad10. c12 = d3 + d4
Zad11. c21 = d5 + d6
Zad12. c22 = d7 + d8

Stopień współbieżności 8 + synchronizacja !

Problemy...?

OpenMP

Podstawy

OpenMP

http://openmp.org/wp/

Realizacja OpenMP

- Wszystkie wątki mają dostęp do wspólnej pamięci i danych dzielonych
- Dane mogę być prywatne i wspólne
- Dane prywatne dostępne są tylko dla wątku właściciela
  - rejestr licznika instrukcji
  - stos
  - pamięć na zm. prywatne)
- Transfer danych odbywa się bardziej przejrzyście niż w MPI
- Synchronizacja jest wciąż potrzebna lecz jest przeważnie ukryta

OpenMP

- Literatura (podstawa opracowania):
  - Hermanns, M. (2002). Parallel Programming in Fortran 95 using OpenMP. http://link.aip.org/link/?CPHYE2/8/117/1
- Dyrektywy
  - Klauzule, warunki
- Funkcje i procedury czasu wykonania
- Zmienne środowiskowe

OpenMP

- Pojęcie dyrektywy OpenMP
- Obszar równoległy
- Warunki wyboru (clause)
- Przykłady programów
- Omówienie podstawowych dyrektyw

Dyrektywy - Fortran

- Dyrektywy OpenMP rozpoczynają się w Fortranie 95 od `!$OMP`; ! oznacza komentarz i dyrektywa jest pomijana w przypadku gdy program chcemy wykonać sekwencyjnie. W C/C++: `#pragma omp ...`
- W starszym wydaniu fortranu mamy: `C$OMP` lub `*$OMP`
- Podobnie `!$` ma specjalne znaczenie w kompilacji równoległej programu - oznacza kompilację warunkową
- Struktura dyrektywy:

```
!$OMP dyrektywa warunki/klauzule
!$OMP end dyrektywa warunki
```

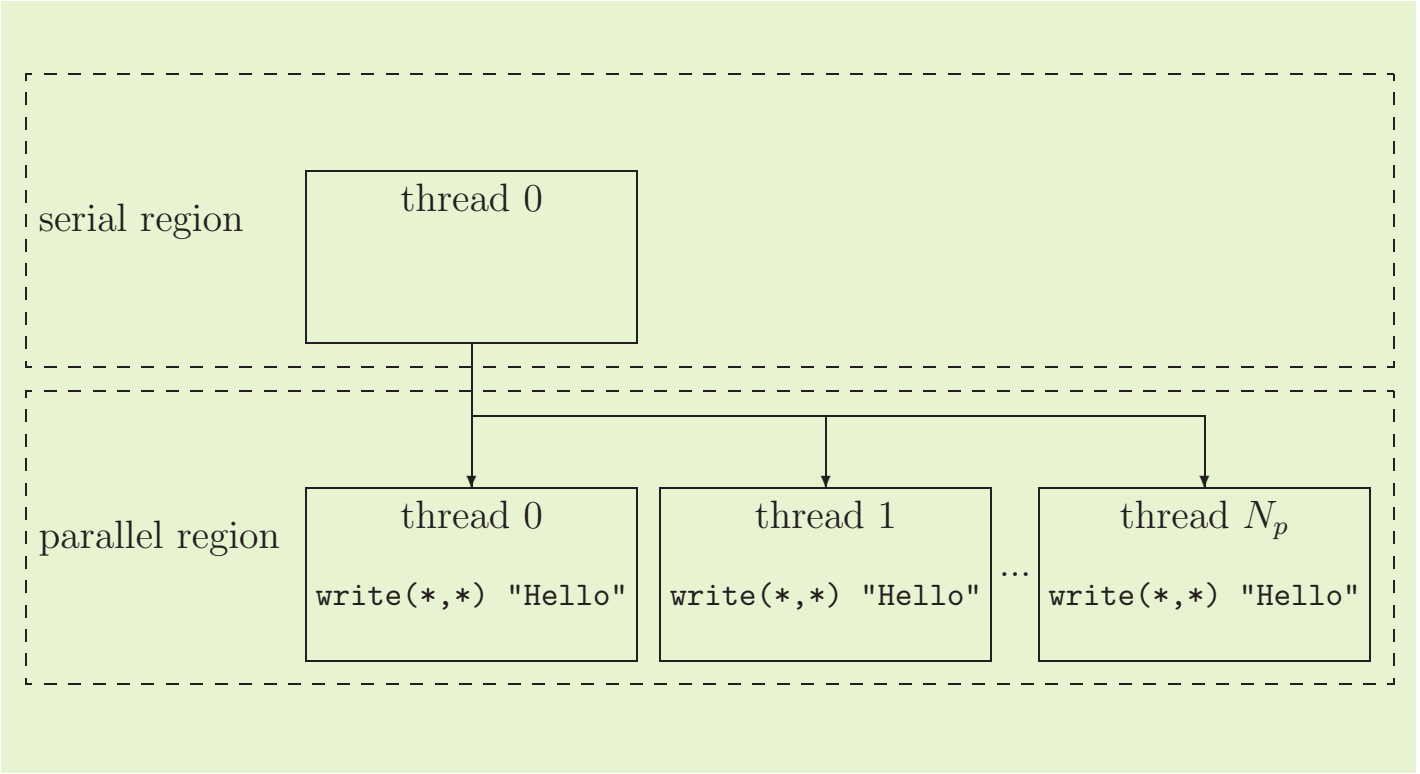
!$omp parallel

- Przyklad prostego konstruktora obszaru równoległego wykonania

```
!$omp parallel
    write(*,*) ‘Hello’
!$omp end parallel
```

Jeśli liczba procesów wynosi np. 4 to słowo „Hello” zostanie wydrukowane 4 razy.

Obszar równoległy



Kompilacja warunkowa

- Przykład kompilacji warunkowej

```
!$ interval = L*OMP_get_thread_num()/ &
!$                (OMP_get_num_threads()-1)
```

Znak `&` (`ampersand`) jest znakiem kontynuacji.

Dyrektywy zagnieżdżone

- Przykład

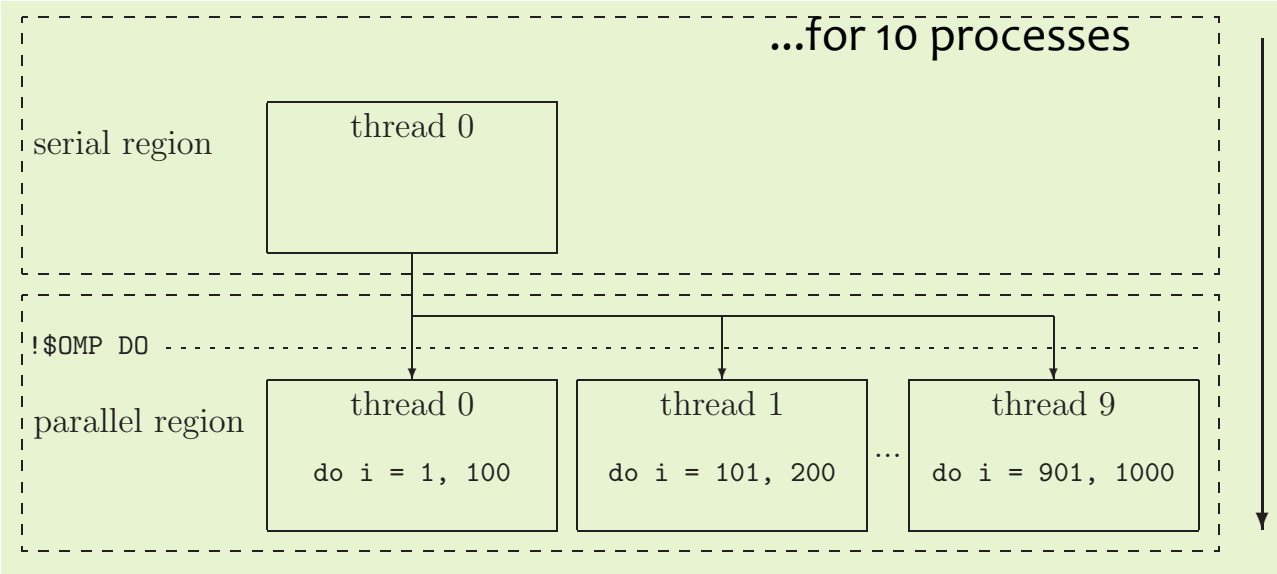
```
!$omp parallel
    write(*,*) ‘Hello’
    !$omp parallel
       write(*,*) ‘Hi!’
    !$omp end parallel
!$omp end parallel
```

Jeśli liczba procesów wynosi N to pojawi się N^2+N inii wyników. Dlaczego?

!$omp do

- Przykład

```
!$omp do
  do i=1, 1000
     ...
  end do
!$omp end do
```



!$omp do

- Niebezpieczeństwa

```
dimension a(100), b(100)
do i=1, 100
   b(i) = 11 * a(i)
   a(i) = a(i) + b(i)
end do
```

Tutaj `b(i)` nie są określone aż do momentu wykonania `!$omp end do`

!$omp do

- Niebezpieczeństwa

```
dimension a(100), b(100)
do i=1, 99
   a(i) = a(i+1)
end do
```

Wyniku działania nie można tutaj przewidzieć.
W chwili wykonywania operacji o numerze `i` element `a(i+1)` nie musi być w stanie pierwotnym – mógł być już zmieniony (*racing*)

… rozwiązanie


```
! MOŻLIWE ROZWIĄZANIE PROBLEMU…
real(8) :: A(1000), dummy(2:1000:2)
!Saves the even indices
!$OMP DO
do i = 2, 1000, 2
      dummy(i) = A(i)
enddo
!$OMP END DO
!Updates even indices from odds
!$OMP DO
do i = 0, 998, 2
      A(i) = A(i+1)
enddo
!$OMP END DO
```

```
!Updates odd indices with evens
!$OMP DO
do i = 1, 999, 2
   A(i) = dummy(i+1)
end do
!$OMP END DO
```

!$omp do (czy optymalnie...?)


```
do i = 1, 10
  do j = 1, 10
    !$omp do
    do k = 1, 10
      A(i, j, k) = i*j*k
    enddo
    !$omp end do
  enddo
enddo
```

```
!$OMP DO
  do i = 1, 10
    do j = 1, 10
      do k = 1, 10
        A(i,j,k) = i * j * k
      enddo
    enddo
  enddo
!$OMP END DO
```

Najlepszym jest rozwiązanie pokazane z lewej strony. Dlaczego?

A jak jest w przypadku C/C++?

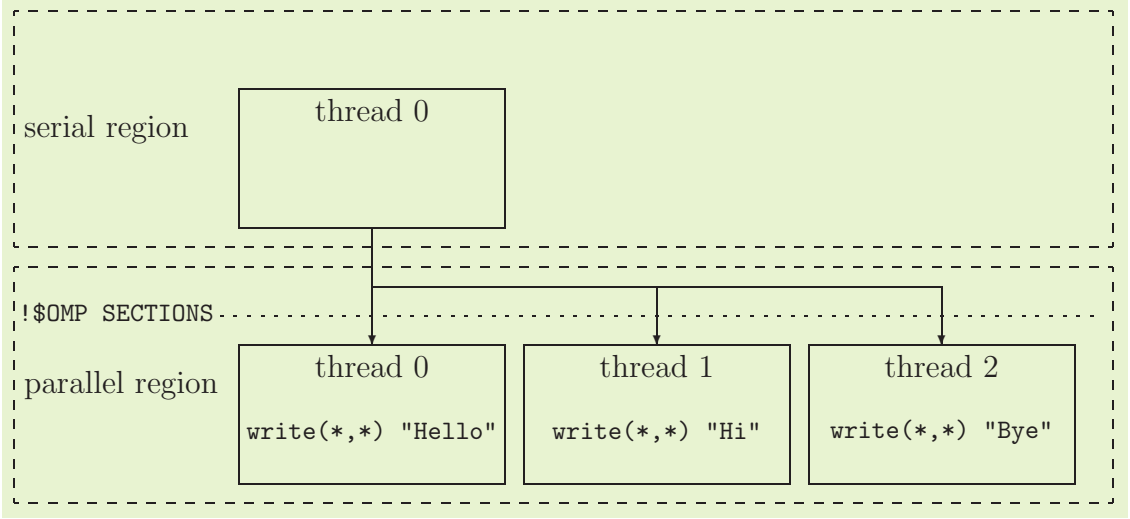
Tablice w fortranie są umieszczane w pamieci komputera „KOLUMNAMI”.

```
!$OMP DO
  do k = 1, 10
    do j = 1, 10
      do i = 1, 10
        A(i,j,k) = i * j * k
      enddo
    enddo
  enddo
!$OMP END DO
```

!$omp sections / ...end...

- ‘`sections`’ używa się tam gdzie mamy niezależne części programu
- Przykład

```
!$OMP SECTIONS
!$OMP SECTION
  write(*,*) "Hello"
!$OMP SECTION
  write(*,*) "Hi"
!$OMP SECTION
  write(*,*) "Bye"
!$OMP END SECTIONS
```



!$omp single / ...end...

- Para `!$omp single` and `!$omp end single` oznacza obszar, w którym może pracować tylko jeden proces, pierwszy, który tam dotrze;

!$omp workshare / ...end...

- Pracę funkcji wewnętrznych Fortranu, które pracują na tablicach, takich jak

  `matmul, dot product, sum, product, maxval, minval, count, any, all, spread, pack, unpack, reshape, transpose, eoshift, cshift, minloc and maxloc`

  można zrównoleglić używając pary dyrektyw:

```
!$omp workshare
!$omp end workshare
```

!$omp workshare / ...end...

33

The Workshare construct

Fortran has a fourth worksharing construct:

```
!$OMP WORKSHARE

    <array syntax>

!$OMP END WORKSHARE [NOWAIT]
```

Example:

```
!$OMP WORKSHARE
    A(1:M) = A(1:M) + B(1:M)
!$OMP END WORKSHARE NOWAIT
```

!$omp workshare / ...end...

Przykład, w którym dyrektywa "DO" nie może być użyta (Dlaczego?):

```
!$OMP DO
   do i = 1, 1000
      B(i) = 10 * i
      A(i) = A(i) + B(i)
    end do
!$OMP END DO
```

...ŹLE

Można zastąpić takim rozwiązaniem:

```
!$OMP WORKSHARE
    forall(i=1:1000)
       B(i) = 10 * i
    end forall
    A = A + B
!$OMP END WORKSHARE
```

DOBRZE...

Kombinacje dyrektyw

- `$OMP PARALLEL DO/!$OMP END PARALLEL DO`
- `!$OMP PARALLEL SECTIONS/!$OMP END PARALLEL SECTIONS`
- `!$OMP PARALLEL WORKSHARE/!$OMP END PARALLEL WORKSHARE`

!$omp master

Para **`!$omp master / !$omp end master`**
pozwala pracować w danym obszarze tylko procesowi głównemu (master process lub root)

!$omp critical / ...end...

- W danym czasie, w obszarze pary dyrektyw „critical” (`!$omp critical/!$omp end critical`) może działać tylko jeden proces (który pierwszy się w nim znajdzie). Ma to zastosowanie w przypadku np. czytania/drukowania (input/output).

```
!$OMP CRITICAL [name]
...
!$OMP END CRITICAL [name]
```

Tutaj `name` jest opcjonalną nazwą dla obszaru krytycznego – sekcje posiadające taką samą nazwą są traktowane jak wspólna sekcja krytyczna.

!$omp barrier

- Dyrektywa „barrier”oznacza miejsce, w którym wymaga się synchronizacji wszystkich procesów (procesy czekają na siebie)
- Należy uważać by nie używać konstrukcji, które prowadzą do śmierci (zakleszczenia; deadlock) programu. Np. konstrukcja

```
!$OMP CRITICAL
   !$OMP BARRIER
!$OMP END CRITICAL
```

prowadzi do sytuacji gdzie jeden wątek czeka na pozostałe, ale te nie mogą wejść do sekcii „critical”

!$omp barrier

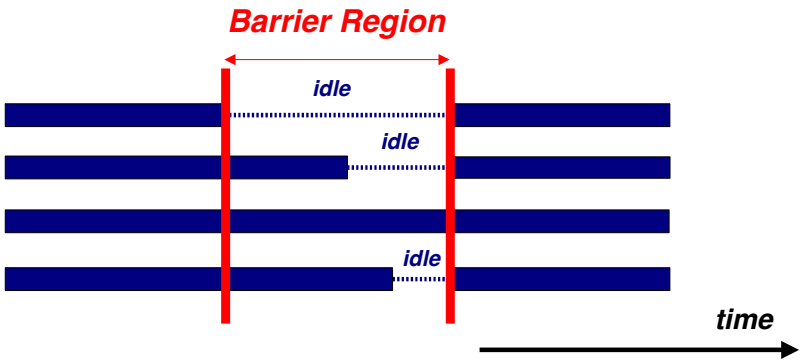
- Inne sytuacje impasu (*deadlock*):

```
!$OMP SINGLE
      !$OMP BARRIER
!$OMP END SINGLE

!$OMP MASTER
      !$OMP BARRIER
!$OMP END MASTER

!$OMP SECTIONS
      !$OMP SECTION
      !$OMP BARRIER
      !$OMP SECTION
      ...
      ...
!$OMP END SECTIONS
```

!$omp barrier

26

Barrier/3



Barrier syntax in OpenMP:

```
#pragma omp barrier
```

```
!$omp barrier
```

!$omp atomic

- Dyrektywa ogranicza dostęp do operacji na obiekcie tylko do jednego wątku w danym czasie. Procesy pojedynczo ją wykonuja. Przykład:

```
!$OMP DO
    do i = 1, 1000
          a = a + I      ! x = x operator expr
    end do
!$OMP END DO
```

Zmienna `a` może być uzupełniana tylko przez jeden proces w danym momencie… `!$omp atomic` może być użyta z operatorami `+, *, -, /, .and., …, Max, Min, Iand, Ior, Ieor.`

!$omp flush

- Używamy w miejscu, w którym wymaga się by wszystkie wątki widziały to samo, te same wartości zmiennych wspólnych (dzielonych przez wątki), w danym momencie.

Zadanie. Opisz dokładnie jak działa i gdzie jest stosowana dyrektywa `!$omp flush`.

Synchronizacja...

- Zazwyczaj synchronizacja odbywa się automatycznie. Jest tak w przypadku dyrektyw:
  - `!$OMP BARRIER`
  - `!$OMP CRITICAL and !$OMP END CRITICAL`
  - `!$OMP END DO`
  - `!$OMP END SECTIONS`
  - `!$OMP END SINGLE`
  - `!$OMP END WORKSHARE`
  - `!$OMP ORDERED and !$OMP END ORDERED`
  - `!$OMP PARALLEL and !$OMP END PARALLEL`
  - `!$OMP PARALLEL DO and !$OMP END PARALLEL DO`
  - `!$OMP PARALLEL SECTIONS and !$OMP END PARALLEL SECTIONS`
  - `!$OMP PARALLEL WORKSHARE and !$OMP END PARALLEL WORKSHARE`

Synchronizacja

- Dyrektywy, które nie wymuszają synchronizacji:
  - `!$OMP DO`
  - `!$OMP MASTER and !$OMP END MASTER`
  - `!$OMP SECTIONS`
  - `!$OMP SINGLE`
  - `!$OMP WORKSHARE`

- Jeśli dodano w dyrektywie warunek (clause) `NOWAIT` wówczas synchronizacja wymuszona zostaje wyłączona!

!$omp ordered/...end...

- Zapewnia prawidłowy porządek wykonania bloku sekwencyjnego wewnątrz instrukcji iteracyjnej DO/for wykonywanej równolegle

```
#pragma omp ordered
```

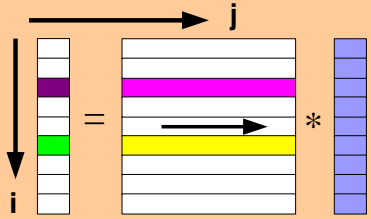
Klauzule, warunki

- Większość dyretyw OpenMP dopuszcza dodatkowe warunki, tzw. klauzule (clauses)
  - Warunki te pozwalają sprecyzować działanie dyrektywy
- Przykładowo `private(a)` jest warunkiem dla dyrektywy `do`
  - `!$omp do private(a)`
- Konkretne warunki, których można używać zależą od dyrektywy

42

Example - Matrix times vector

```
#pragma omp parallel for default(none) \
            private(i,j,sum) shared(m,n,a,b,c)
for (i=0; i<m; i++)
{
   sum = 0.0;
   for (j=0; j<n; j++)
     sum += b[i][j]*c[j];
   a[i] = sum;
 }
```



TID = 0

```
for (i=0,1,2,3,4)
 i = 0
   sum =   b[i=0][j]*c[j]
   a[0] = sum
 i = 1
   sum =   b[i=1][j]*c[j]
   a[1] = sum
```

TID = 1

```
for (i=5,6,7,8,9)
 i = 5
   sum =   b[i=5][j]*c[j]
   a[5] = sum
 i = 6
   sum =   b[i=6][j]*c[j]
   a[6] = sum
```

... etc ...

Podsumowanie

- Krótkie podsumowanie OpenMP (pliki pdf)
  - http://www.openmp.org/mp-documents/OpenMP3.0-FortranCard.pdf
  - http://www.openmp.org/mp-documents/OpenMP3.0-SummarySpec.pdf
- Dobry przegląd dyrektyw OpenMP można znaleźć na stronie Sun Microsystems (OpenMP API User's Guide):
  - http://docs.sun.com/source/819-0501/index.html

CUDA

Informacja

CUDA

- CUDA – Compute Unified Device Architecture
  - NVIDIA, GPU
  - Narzędzia do przygotowywania programów w C/C++
  - Biblioteki funkcji (**`math.h`**, FFT, BLAS)
  - Nie jest konieczne szczegółowe zapisywanie wątków i zarządzanie nimi (jak np. w OpenMP)
  - Program zawiera fragmenty sekwencyjne, wykonywane przez CPU oraz fragmenty równoległe, do wykonania w GPU
  - Bloki wątków – 1, 2, 3 wymiarowe tablice wątków (tzw. *gridy*)

CUDA - przykład

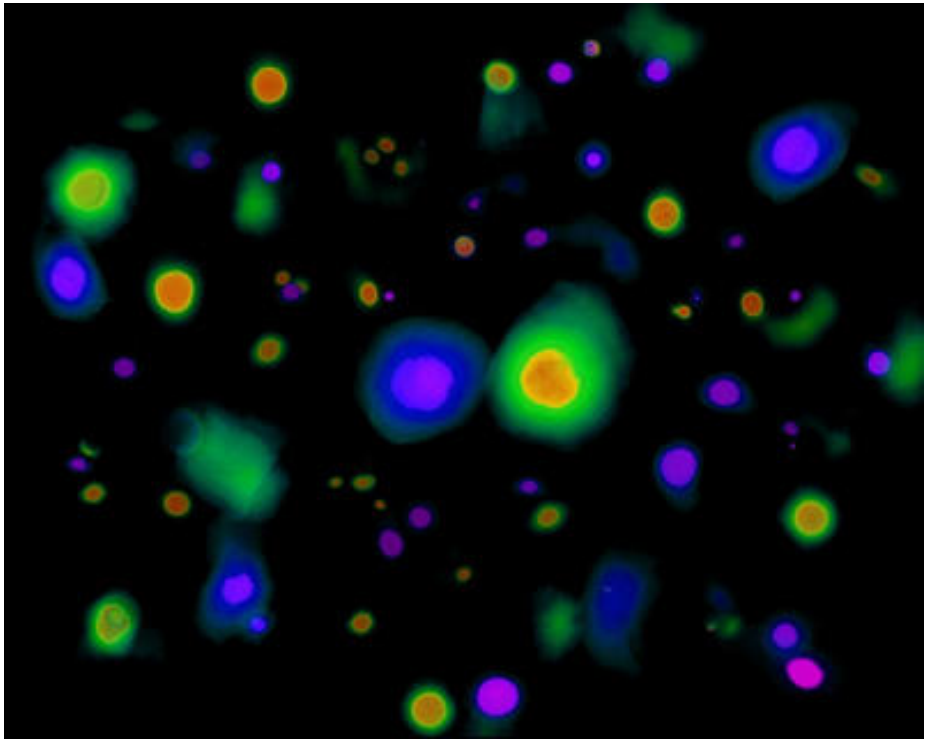
- Suma wektorów y = alfa * x + y;
  - **OBLICZENIA SEKWENCYJNE**

```
void saxpy_sekw(int n, float alfa, float *x, float *y)
{
   for (int i=0; i<n; i++){
      y[i] = alfa * x[i] + y[i]
}
/* wywołanie funkcji saxpy */
saxpy_sekw(n, 2.3, x, y);
```

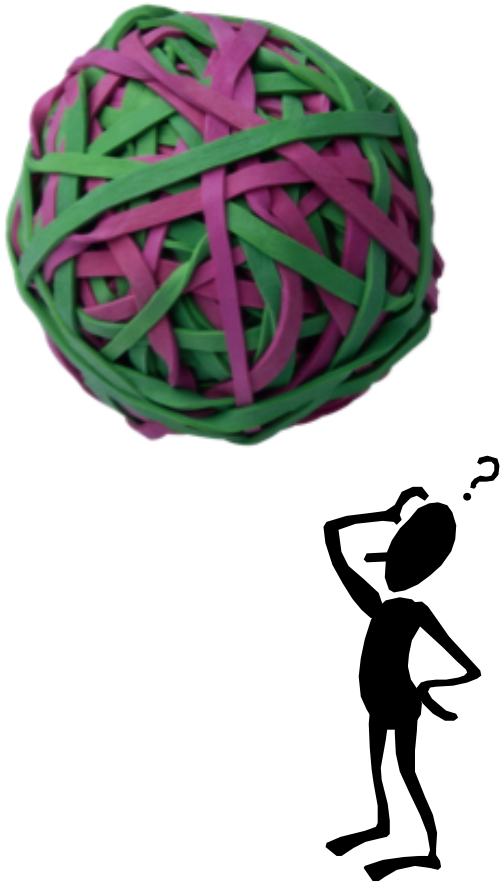
  - **OBLICZENIA RÓWNOLEGLE**

```
__global__ void saxpy_par(int n, float alfa, float *x, float *y)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
if (i<n) y[i] = alfa * x[i] + y[i];
}
/* wywołanie jądra saxpy_par */
int liczba_blokow = (n+256) / 256;
saxpy_par<<<liczba_blokow, 256>>>(n, 2.3, x, y);
```

Problemy...?



A random collection of textures taken from high-resolution, supercomputer simulations. Red indicates a positive twist in the topological charge density and blue a negative twist. (Credit: V. Travieso and N. Turok)



I TO BY BYŁO NA TYLE!

- DZIĘKUJĘ PAŃSTWU ZA WSPÓŁPRACĘ NA WYKŁADZIE, ZA UWAGI I PYTANIA
- ZAPRASZAM NA EGZAMIN!
- TERMIN: … ?