OpenMP

Programowanie współbieżne... (9)

Andrzej Baran 2010/11

LINK: http://kft.umcs.lublin.pl/baran/prir/index.html

Powtórzenie: klauzule...

KLAUZULA / pragma	PARALLEL	DO /for	SECTIONS	SINGLE	PARALLEL DO / for	PARALLEL SECTIONS	PARALLEL WORKSHARE
if	X				X	X	X
private	X	X	X	X	X	X	X
shared	X				X	X	X
firstprivate	X	X	X	X	X	X	X
lastprivate		X	X		X	X	
default	X				X	X	X
reduction	X	X	X		X	X	X
copyin	X				X	X	X
copyprivate				X			
ordered		X			X		
schedule		X			X		
nowait		X	X	X			
num_threads	X				X	X	X

Literatura (podstawa opracowania):

Hermanns, M. (2002). Parallel Programming in Fortran 95 using OpenMP. [pdf] http://link.aip.org/link/?CPHYE2/8/117/1

Treść

- **Wstęp**
- **Dyrektywy**
  - **Klauzule, warunki**
- **Funkcje i procedury czasu wykonania**
- **Zmienne środowiskowe**

Biblioteka czasu wykonania

- **OpenMP Fortran API run-time library – zawiera procedury pozwalające kontrolować wykonanie programu oraz uzyskiwać informacje o środowisku**
- **Fortran: biblioteka ta znajduje się w module omp_lib**

- PROCEDURY/FUNKCJE
  - **OMP_set_num_threads, OMP_get_num_threads, OMP_get_max_threads, OMP_get_thread_num, OMP_get_num_procs, OMP_in_parallel, OMP_set_dynamic, OMP_get_dynamic, OMP_set_nested, OMP_get_nested**

Procedury run-time

Jako przykład, zobaczmy nagłówek procedury
OMP_set_num_threads w jej deklaracji w interfejsie.

```
subroutine OMP_set_num_threads(number_of_threads)
integer(kind = OMP_integer_kind),  &
            intent(in) :: number_of_threads
end subroutine OMP_set_num_threads
```

Zmienna **number_of_threads** jest liczbą wątków, które chcemy uruchomić (w przypadku dynamicznym)

Procedury run-time

- **OMP_get_max_threads** – maksymalna liczba wątków, możliwych w programie
- **OMP_get_thread_num** – numer aktualnego wątku (0 = master)
- **OMP_get_num_procs** – liczba dostępnych procesorów
- **OMP_in_parallel** – **.true.** jeśli wywołanie jest w obszarze równoległym
- **OMP_set_dynamic** – procedura pozwala ustawić możliwość dynamicznego zarządzania liczbą wątków

```
subroutine OMP_set_dynamic(enable)
        logical(kind = OMP_logical_kind), intent(in) :: enable
end subroutine OMP_set_dynamic
```

Zmienna enable może przyjmować wartości **.true.** i **.false.**

Procedury run-time

- **OMP_get_dynamic** – zwraca status możliwości dynamicznego zarządzania liczbą wątków (**.true., .false.**)
- **OMP_set_nested(enable)** – zezwala lub uniemożliwia zagnieżdżoną współbieżność (w zależności od wartości zmiennej **enable**)
- **OMP_set_nested()** – sprawdza status zagnieżdżania; wynik: **.true.** lub **.false.**

Procedury blokujące

- Wykluczanie zapewnia bezpieczeństwo wątku. Efekt ten uzyskuje się stosując zamki (zamek=lock). Tylko wątek będący właścicielem zamka (kontrolowana zmienna) może operować w danym obszarze programu. Inne wątki z obszaru muszą czekać do chwili gdy któryś z nich stanie się właścicielem tego zamka po uprzednim zwolnieniu go przez inny wątek.
- Funkcjami (run-time w OpenMP), które zarządzają zamkami i informują o ich stanie są funkcje typu „**..._lock_...**”: **OMP_init_lock, OMP_set_lock, OMP_unset_lock, OMP_test_lock, OMP_destroy_lock**
- Zapoznamy się z podstawami ich użycia na przykładach

Przykład

- Załóżmy, że każdy z pracujących w danym obszarze, czterech wątków, może modyfikować tablicę A(1:100, 1:100). Porządek nie gra roli. Aby uniknąć niekorzystnej sytuacji wyścigu do danych można zastosować następujące rozwiązanie

```
!$OMP PARALLEL SHARED
!$OMP CRITICAL
...            ! Praca z A
!$OMP END CRITICAL
!$OMP END PARALLEL
```

Cała tablica A należy do jednego wątku. Lepsze rozwiązanie polega na podziale tablicy na cztery części A1=A(1:50,1:50), A2=A(51:100,1:50), ... i na wykorzystaniu pozostałych wątków. Każdy wątek może zmieniać wszystkie cztery bloki A1,...,A4 i najlepszą będzie sytuacja gdy będą to robić razem, jeden wątek – jeden blok. W ten sposób eliminujemy wyścig i zyskujemy na czasie. Taką synchronizację można otrzymać stosując zamki.

Zamki – Jak?

Jak to robimy w OpenMP?

① Inicjujemy zamek (lock) i odpowiadający mu identyfikator
② Wątek zostaje właścicielem zamka
③ Inne wątki, zainteresowane, sprawdzają stan zamka
④ Po zwolnieniu, zamek przechwytuje inny właściciel
⑤ Po wykonaniu zadania zamek zostaje zniesiony

Przykład

```
program Main  ! Hermanns
    use omp_lib
    implicit none
    integer(kind = OMP_lock_kind) :: lck
    integer(kind = OMP_integer_kind) :: ID
    call OMP_init_lock(lck)           ! Inicjalizacja zamka
    !$OMP PARALLEL SHARED(LCK) PRIVATE(ID)
      ID = OMP_get_thread_num()
      call OMP_set_lock(lck)          ! Zamek ustawiony
      write(*,*) "My thread is ", ID  ! Tylko właściciel lck
      call OMP_unset_lock(lck)        ! Zamek zniesiony
    !$OMP END PARALLEL
    call OMP_destroy_lock(lck)        ! Zamek zniszczony
end program Main
```

Procedury pomiaru czasu

Program poprzedni można zastąpić programem, który używa dyrektywy OpenMP: !OMP_CRITICAL (*Hermanns*)

```
program Main
  use omp_lib
  implicit none
  integer(kind = OMP_integer_kind) :: ID
  !$OMP PARALLEL SHARED(LCK) PRIVATE(ID)
    ID = OMP_get_thread_num()
  !$OMP CRITICAL
    write(*,*) "My thread is ", ID
  !$OMP END CRITICAL
  !$OMP END PARALLEL
end program Main
```

Przykład

Nie zawsze można zastąpić procedury “lock” kombinacją dyrektyw OpenMP. Pokazuje to następny przykład (*Hermanns*).

```
program Main
use omp_lib
implicit none
integer(kind = OMP_lock_kind) :: lck
integer(kind = OMP_integer_kind) :: ID
call OMP_init_lock(lck)
!$OMP PARALLEL SHARED(LCK) PRIVATE(ID)
   ID = OMP_get_thread_num()
   do while(.not.OMP_test_lock(lck))
   ... ! Proces, który czeka by stać się właścicielem zamka,
... ! wykonuje jakąś prace
   enddo
   ... ! Praca wykonywana przez właściciela zamka
!$OMP END PARALLEL
call OMP_destroy_lock(lck)
end program Main
```

Tutaj, procesy, które nie są właścicielami zamka, nie są bezczynne - robią coś innego. Nie da się tego osiągnąć, używając tylko dyrektyw OpenMP.

Pomiar czasu

Pomiar czasu odbywa si z pomocą procedur

OMP_get_wtime – czas zegara (ściennego)

OMP_get_wtick – czas, w sekundach, między kolejnymi tyknięciami zegara; precyzja czasomierza.

Przykład. Pomiar czasu obliczeń.

```
start = OMP_get_wtime()
... ! Fragment programu gdzie mierzymy czas
end = OMP_get_wtime()
time = end - start
```

Zmienne środowiskowe

- Zmienne środowiskowe organizują środowisko pracy OpenMP w danym systemie
- Zmienne
  - OMP_NUM_THREADS
  - OMP_SCHEDULE
  - OMP_DYNAMIC
  - OMP_NESTED

Biblioteka Gnu...

- GNU libgomp + opis dyrektyw, funkcji run-time, zmiennych

Problemy...?