
Programowanie systemów z pamięcią wspólną - specyfikacja OpenMP

OpenMP – dyrektywy podziału pracy

- wszystkie wątki w zespole muszą realizować te same dyrektywy podziału pracy (i bariery)
- nie ma niejawnej bariery przy wejściu
- rozpoczynanie i kończenie wykonania wybranych dyrektyw może być związane z synchronizacją (realizacją bariery)
- wątki dzielą się pracą – każdy realizuje przydzielone sobie operacje na przydzielonej sobie części danych

OpenMP – dyrektywy podziału pracy

→ **for**

`#pragma omp for lista_klauzul`

`for(.....){.....}` // pętla w postaci kanonicznej

▪ ***lista_klauzul***:

- ♦ *klauzule_zmiennych* (`private`, `firstprivate`, `lastprivate`, `reduction`)
- ♦ **`schedule`**(*rodzaj*, *rozmiar_porcji*) ; rodzaje: `static`, `dynamic`, `guided`, `runtime`
- ♦ inne: `nowait`, `ordered`

▪ obszar równoległy składający się z pojedynczej równoległej pętli **for**:

`#pragma omp parallel for lista_klauzul znak_nowej_linii for(.....)`
`{.....}`

OpenMP – przykład

```
#define LICZBA 100
int main( int argc, char *argv[] ){
    int i; int suma=0;
    // alternatywa dla użycia klauzuli reduction
#pragma omp parallel default(none) shared(suma)
{
    int suma_tmp= 0;
#pragma omp for
    for( i=0; i<=LICZBA; i++){ suma_tmp += i; }
#pragma omp critical(suma) // lub #pragma omp atomic
    suma += suma_tmp;
#pragma omp barrier
    printf(„Suma końcowa %d, mój udział %d\\n”, suma, suma_tmp);
}
}
```

OpenMP – redukcja

→ **reduction(*op*: *lista_zmiennych*)**

```
#pragma omp parallel for reduction(+:a)
```

```
for(i=0;i<n;i++) a += b[i];
```

- możliwe operatory (**op**): +, *, -, &, ^, |
- zmienna redukowana nie może być prywatna przed klauzulą **reduction**
- na początku pętli dla każdego wątku tworzona i odpowiednio inicjowana jest prywatna kopia zmiennej redukowanej (o tej samej nazwie)
- wynik po zakończeniu pętli równoległej, będący efektem uaktualnienia oryginalnej zmiennej redukowanej za pomocą operacji **op** i prywatnych kopii zmiennej dla każdego z wątków, ma być taki sam jak byłby po wykonaniu sekwencyjnym
- pętla nie może kończyć się poprzez **break**
- zmiana parametrów petli (w tym wartości zmiennej sterującej) jest niedozwolona w treści iteracji
- szczegóły realizacji są pozostawione implementacji

OpenMP – przykład

```
#include <omp.h>

#define LICZBA 100
#define LICZBA_W 4

int main( int argc, char *argv[] ){
    int i; int suma=0;
    #pragma omp parallel for reduction(+:suma) num_threads(LICZBA_W)
    for( i=0; i<=LICZBA; i++){ suma += i; }
}
```

OpenMP – dyrektywy podziału pracy

→ single

```
#pragma omp single lista_klauzul  
{ .... }
```

▪ *lista_klauzul*:

- ◆ *klauzule_zmiennych* (*private*, *firstprivate*, *copyprivate*)
- ◆ *nowait*

→ master

```
#pragma omp master znak_nowej_linii {....}
```

- brak klauzul, brak bariery na zakończenie

OpenMP – dyrektywy podziału pracy

→ sections

```
#pragma omp sections lista_klauzul
{
    #pragma omp section
    {...}
    #pragma omp section
    {...}
    /* itd. */
}
```

- *lista_klauzul*: private, firstprivate, lastprivate, reduction, nowait
- obszar równoległy składający się z pojedynczej struktury sections:
#pragma omp parallel sections *itd.*

OpenMP – funkcje biblioteczne

→ funkcje związane ze środowiskiem wykonania:

- plik nagłówkowy: `omp.h`
- składnia funkcji set: `void funkcja(int);`
- składnia pozostałych funkcji: `int funkcja(void)`
- `omp_set_num_threads` - ustalenie liczby wątków
- `omp_get_num_threads` - pobranie liczby wątków
- `omp_get_num_procs` - pobranie liczby procesorów
- `omp_get_thread_num` - pobranie rangi konkretnego wątku (master - 0)
- `omp_in_parallel` - sprawdzenie wykonania równoległego
- `omp_get_max_threads` - pobranie maksymalnej liczby wątków
- `omp_set_dynamic, omp_get_dynamic` – dostosowywanie liczby wątków
- `omp_set_nested, omp_get_nested` – umożliwianie zagnieżdżania

OpenMP – funkcje biblioteczne

→ funkcje obsługi zamków:

- typ zamka: `omp_lock_t`; argumentem funkcji jest zawsze `omp_lock_t*`
- `omp_init_lock` - inicjowanie
- `omp_destroy_lock` - niszczenie
- `omp_set_lock` - zamykanie
- `omp_test_lock` - próba zamykania bez blokowania
- `omp_unset_lock` - otwieranie
- wersje dla zagnieżdżonych zamków

→ funkcje pomiaru czasu:

- `omp_get_wtime` – czas zegara
- `omp_get_wtick` – rozdzielczość (dokładność) zegara

OpenMP – makro preprocesora i zmienne środowiskowe

```
#ifdef _OPENMP
```

```
    printf(„Kompilator rozpoznaje dyrektywy OpenMP\n”);
```

```
#endif
```

→ Zmienne środowiskowe

- **OMP_SCHEDULE** *określenie* – dla równoległych pętli z klauzulą **schedule(runtime)**
- **OMP_NUM_THREADS** *liczba*
- **OMP_DYNAMIC** *TRUE/FALSE*
- **OMP_NESTED** *TRUE/FALSE*

OpenMP - przykład

```
#include<omp.h>
int main(){
#ifdef  _OPENMP
printf("Kompilator rozpoznaje dyrektywy OpenMP\n");
#endif
int lwat; printf("maksymalna liczba watkow - "); scanf("%d",&lwat);
omp_set_num_threads(lwat);
printf("aktualna liczba watkow %d, moj ID %d\n",
        omp_get_num_threads(), omp_get_thread_num());
#pragma omp parallel
{
printf("aktualna liczba watkow %d, moj ID %d\n",
        omp_get_num_threads(), omp_get_thread_num());
}
```

OpenMP – równoległość zadań

→ task

`#pragma omp task lista_klauzul znak_nowej_linii { ... }`

- generowanie zadań do wykonania przez wątki, realizacja odbywa się w sposób asynchroniczny – zadanie może zostać wykonane od razu, może też zostać zrealizowane w późniejszym czasie przez dowolny wątek z danego obszaru równoległego
- *lista_klauzul*:
 - *klauzule_zmiennych* (`default`, `private`, `firstprivate`, `shared`)
 - `if` – jeśli warunek nie jest spełniony, zadanie nie jest zlecane do wykonania asynchronicznego, jest realizowane od razu
 - `untied` – w przypadku wstrzymania wykonywania może być kontynuowana przez inny wątek niż dotychczasowy
 - `final` – jeśli warunek prawdziwy kolejne napotkane zadania są wykonywane od razu
 - `mergeable` – dla zadań zagnieżdżonych,

OpenMP – równoległość zadań

- Wykonywanie zadań może być przerywane w określonych punktach (*task scheduling points*) i wznowiane zgodnie z zasadami planowania zadań
- Dyrektywy synchronizacji i planowania zadań:
- **taskyield**
 - `#pragma omp taskyield znak_nowej_linii`
 - oznaczenie miejsca, w którym wykonanie zadania może zostać wstrzymane w celu uruchomienia innego zadania
- **taskwait**
 - `#pragma omp taskwait znak_nowej_linii`
 - oczekiwanie na zakończenie wykonywania utworzonych zadań
- dyrektywa **barrier** wymusza dokończenie wykonywania wszystkich zadań utworzonych przed jej wystąpieniem

OpenMP – równoległość zadań

→ Przykład – przetwarzanie listy powiązanej:

```
void process_list(node * head)
{
    #pragma omp parallel default(none) firstprivate(head)
    {
        #pragma omp single
        {
            node * p = head;
            while (p) {
                #pragma omp task default(none) firstprivate(p)
                {
                    process(p);
                }
                p = p->next;
            }
        }
    }
}
```

OpenMP - uwagi końcowe

- specyfikacja OpenMP określa sposób realizacji dyrektyw i funkcji, który jednak nie gwarantuje poprawności programów (w przeciwieństwie do kompilatorów z automatycznym zrównolegleniem, które wprowadzają wykonanie współbieżne tylko tam, gdzie potrafią zagwarantować prawidłowość realizacji)
- programy równoległe OpenMP z założenia są przystosowane także do wykonania sekwencyjnego (przy ignorowaniu dyrektyw i wykonywaniu pustych namiastek funkcji)