OpenMP

Wprowadzenie

Standard OpenMP

- OpenMP (*Open Multi-Processing) jest standardem definiującym*

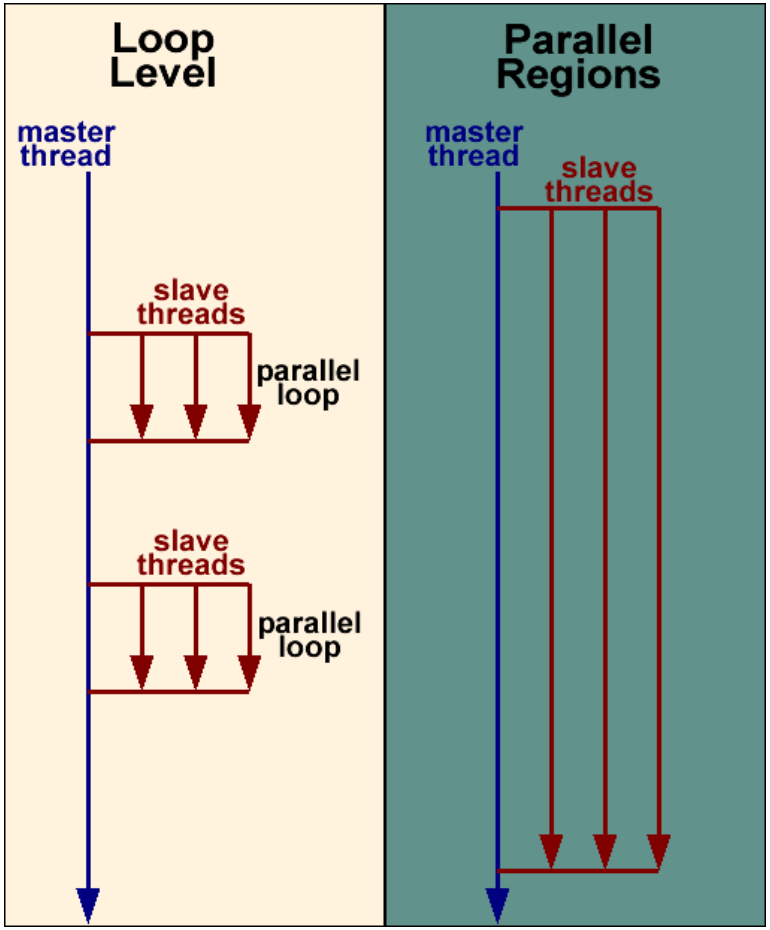
  interfejs programowania aplikacji (API - Application Programming Interface)

  do tworzenia programów równoległych opierających się na modelu wielowątkowym dla systemów z pamięcią wspólną.
- W skład komitetu pracującego nad standardem OpenMP ARB (The OpenMP Architecture Review Board) wchodzą między innymi:

  Sun, NASA, Intel, Fujitsu, IBM, AMD, Cray, HP, SGI, NEC, Microsoft.
- Implementacje znajdują się na wielu platformach sprzętowych zawierających większość systemów UNIX'owych i Windows NT.
- Standard OpenMP - pierwsza wersja 1997 rok dla Fortranu; 1998 rok dla C/C++ - jest stale rozwijany – aktualną wersją standardu jest 3.0.
- Strona domowa: **www.openmp.org**

Standard OpenMP

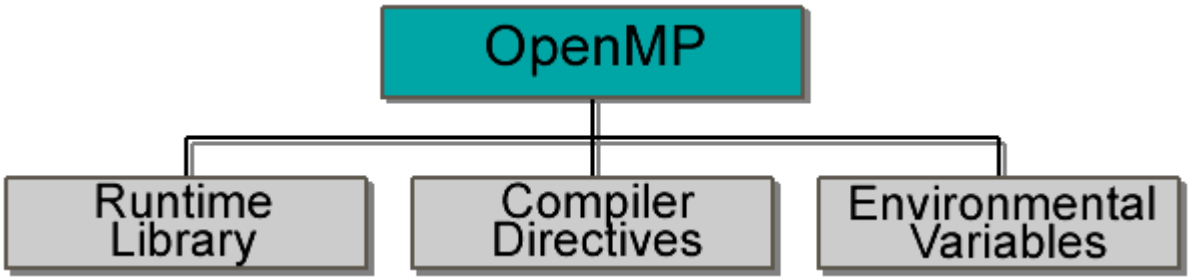
- Dopuszcza zrównoleglenie na poziomie zadań (ang. *task parallelism*), oraz na poziomie danych (ang. *data parallelism*).



Standard OpenMP

- Za przydział wątków do poszczególnych procesorów odpowiada środowisko uruchomieniowe, które stosuje algorytm uwzględniający m.in. aktualne obciążenie poszczególnych procesorów oraz całej maszyny.
- Liczba nowych wątków może być określona tuż przed uruchomieniem programu za pośrednictwem zmiennych środowiskowych lub w kodzie źródłowym za pomocą specjalnych funkcji.
- Deklaracje funkcji biblioteki OpenMP dla języków C/C++ znajdują się w pliku nagłówkowym "omp.h”.

Standard OpenMP



- Składa się z:
  - funkcji bibliotecznych, które umożliwiają ustawienie i zapytanie o parametry przetwarzania równoległego,
  - dyrektyw kompilatora, używanych przez programistę do komunikacji z kompilatorem,
  - zmiennych środowiskowych, , które mogą być użyte do zdefiniowania ograniczonej liczby równoległych parametrów systemu wykonawczego.

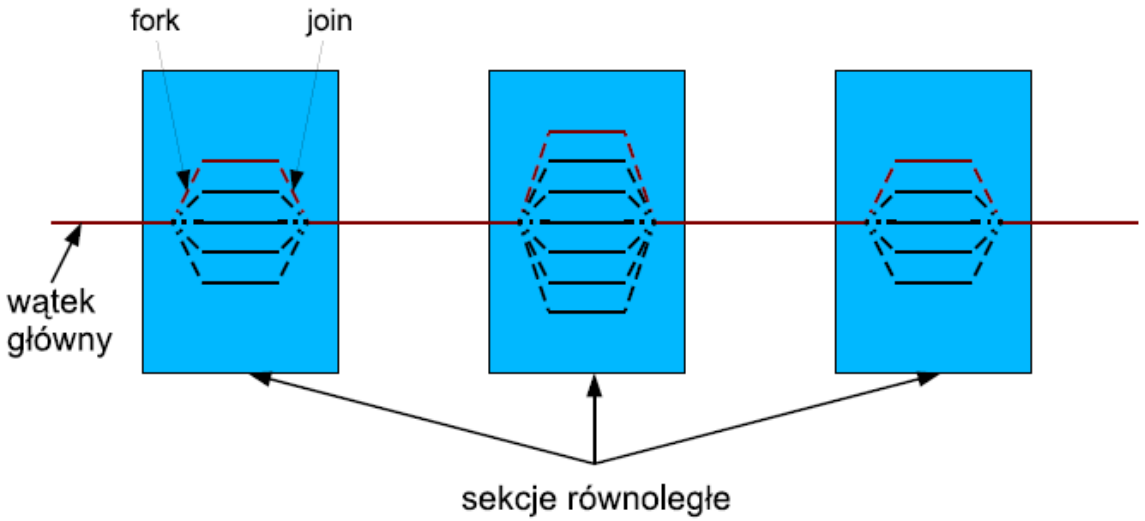
Dlaczego OpenMP jest tak popularny?

- brak przesyłania komunikatów,
- dyrektywy OpenMP lub biblioteki mogą być włączane stopniowo,
- kod jest w istocie kodem sekwencyjnym,
- niewielki wzrost wielkości kodu w stosunku do kodu sekwencyjnego.

Model programowania

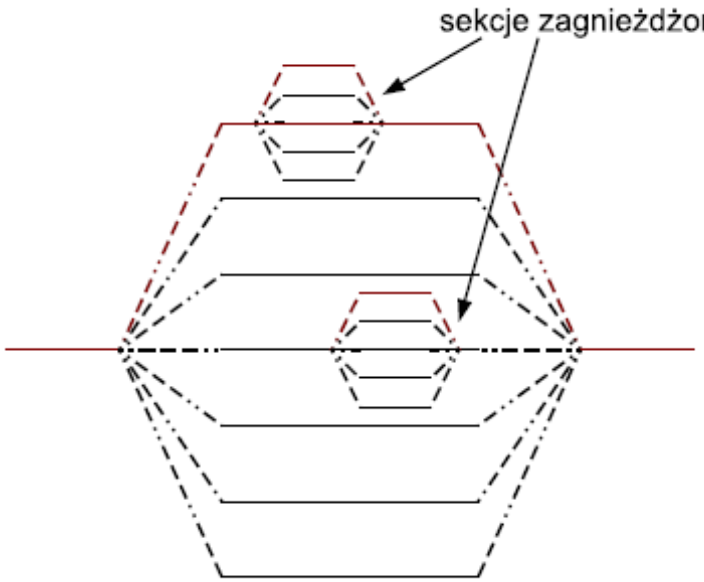
- Zrównoleglanie kodu opiera się o model *fork-and-join:*

  wątek główny tworzy dodatkowe wątki w momencie wykonywania obliczeń równoległych:



Zrównoleglenie zagłębione

- Specyfikacja OpenMP przewiduje zrównoleglenie zagł ębione (*nested parallelism)*.
- Każdy wątek w grupie może utworzyć własną grupę wątków wykonujących się równolegle.
- Aby taki model przetwarzania był możliwy należy ustawić odpowiednią zmienną systemową.



Kompilacja

- #include <omp.h>
- g++ -fopenmp [...]
- icc -fopenmp [...]

#pragma omp parallel

- Dyrektywa **omp parallel** określa obszar wykonywany równolegle przez wątki.

  #pragma omp parallel [klauzula [klauzula]…]

  < C/C++ blok strukturalny >

  Przykład:

```
double a[4];
omp_set_num_threads(4);
#pragma omp parallel
{
    int  id = omp_get_thread_num();
    a[id]=id;
}
```

Zakres widoczności zmiennych

Klauzule:

- **private(list)**

  zmienne są prywatne dla każdego wątku. Ich wartość jest nieokreślona przy wejściu do bloku równoległego i po wyjściu z niego.

- **shared(list)**

  zmienne wymienione w klauzuli są wspólne dla wszystkich wątków.

- **default(shared | none)**

  shared - każda z widocznych zmiennych będzie traktowana jako wspólna, poza zadeklarowanymi przy pomocy threadprivate i stałymi.

  none - wymaga, aby każda zmienna była jawnie zadeklarowana.

- **firstprivate(list)**

  zmienna prywatna, która przy wejściu do bloku równoległego jest kopią zmiennej globalnej.

Przykład:

```
double a = 1;
double b = 2;
double c = 3;
#pragma omp parallel private(b) firstprivate(c)
  {
     (1)
  }
 (2)
```

wewnątrz obszaru równoległego (1):

a jest zmienną wspólną dla wszystkich wątków (jest równa 1),

b i c są zmiennymi lokalnymi dla wątków (wartość początkowa zmiennej b jest niezdefiniowana, natomiast zmiennej c jest równa 3.

poza obszarem równoległym (2):

wartości zmiennych b i c są niezdefiniowane.

#pragma omp threadprivate (list)

oznacza, że wszystkie zmienne podane jako parametry na liście będą prywatne dla wątków w całej przestrzeni programu.

Zrównoleglanie pętli

#pragma omp for [Klauzule]

<pętla języka C/C++, która będzie wykonywana równolegle>

Klauzule:

- private(list)
- firstprivate(list)
- lastprivate(list)

  zmienna globalna, która jest zmienną prywatną podczas wykonywania pętli po jej zakończeniu przyjmuje wartość dla ostatniej iteracji pętli
- reduction(operator: list)
- schedule(kind[, chunk])
- ordered
- nowait

#pragma omp for schedule(rodzaj [,rozmiar_segmentu])

- Dyrektywa schedule służy do definiowania sposobu rozdziału pracy na wątki w pętli for.
- Możliwe rodzaje podziału pracy:
  - static – podział dokonany przed uruchomieniem pętli, najmniejszy narzut czasu wykonania
  - dynamic – wątki wykonują kolejno „pierwszy wolny" segment w przestrzeni instrukcji for
  - guided – trochę jak w dynamic, wielkość segmentu może ulegać zmniejszaniu
  - runtime – podział zależy od wartości zmiennej środowiskowej OMP SCHEDULE

Blok sections

- Dyrektywa sections wyodrębniania fragmenty kodu wykonywane jednocześnie przez różne wątki
- Poszczególne zadania umieszcza sie w blokach section

```
#pragma omp parallel sections shared(x,y)
{
# pragma omp section
    f(x);
# pragma omp section
    g(x);
# pragma omp section
    h(x, x);
# pragma omp section
    j(x, 1);
}
```

Jak zrównoleglać sumy, iloczyny itp?

- Opcja reduction służy do definiowania zmiennych jako wyników sum, iloczynów etc.

```
double suma_dobra (int n)
{
double suma = 0;
int i;
# pragma omp parallel for \
default(none) shared(n) private(i), reduction(+ : suma)
for (i = 2; i <= n; i++)
    suma += sqrt(2*i/0.887)+pow(exp(3*i),6);
return suma;
}
```

Synchronizacja watków

Do synchronizacji lub desynchronizacji pracy wątków służą m.in. dyrektywy:

- barrier
- critical
- master
- atomic

i opcje:

- nowait

Dyrektywa barrier

- Dyrektywa barrier wstrzymuje wątki, aż wszystkie wątki zespołu osiagną barierę

```
# pragma omp parallel
{
...
# pragma omp barrier
...
}
```

Bariery domyślnie ustawiane są na końcu bloków instrukcji objętych niektórymi dyrektywami, m.in. for, sections i single.

Opcja nowait

- Opcja nowait wyłącza domyślną barierę

```
# pragma omp parallel
{
...
# pragma omp single nowait
...
# pragma omp for nowait
...
# pragma omp sections nowait
...
}
```

Dyrektywa critical

- Dyrektywa critical tworzy region („sekcje krytyczna”), który może być wykonywany przez co najwyżej jeden wątek naraz
- Powinna zawierać kod, który wykonuje sie szybko, by nie blokować pracy innych wątków

Zmienne środowiskowe

- **OMP_SHEDULE** - ustawia rodzaj i ewentualnie porcje dla parametru runtime klauzuli shedule
- **OMP_NUM_THREADS -** ustawia ilość wątków wykorzystywaną podczas wykonywania programu.
- **OMP_DYNAMIC -** ustawia lub blokuje dynamiczne przydzielanie wąków.
- **OMP_NESTED -** ustawia lub blokuje zagnieżdżanie równoległości.

Funkcje OpenMP

- void omp_set_num_threads (int); // ustaw # wątkow w nast. parallel
- int omp_get_num_threads (void); // ile aktywnych wątkow?
- int omp_get_max_threads (void);
- int omp_get_thread_num (void); // numer watku
- int omp_get_num_procs (void); // liczba procesorow
- double omp_get_wtime (void);
- double omp_get_wtick (void);
- int omp_in_parallel (void);
- void omp_set_dynamic (int);
- int omp_get_dynamic (void);
- void omp_set_schedule (omp_sched_t, int);
- void omp_get_schedule (omp_sched_t *, int *);

Literatura

- *OpenMP Application Program Interface Version 3.0 May 2008*
- Zbigniew Koza, *OpenMP-część praktyczna,*
- Tomasz Olas, *Wprowadzenie do zrównoleglania aplikacji z wykorzystaniem standardu OpenMP.*