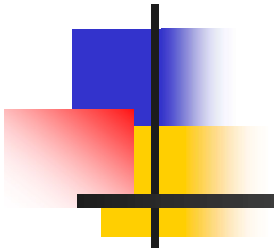


Calcolo Parallelo e Distribuito

openMP:

il costrutto FOR della classe dei costrutti WorkSharing

ulteriori osservazioni

Docente: Prof. L. Marcellino

Tutor: Prof. P. De Luca

… ancora qualche osservazione sulla combinazione dei costrutti parallel e for

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main( )
{
   int n_threads, id_thread, i;

   omp_set_num_threads(3);
   #pragma omp parallel for private(id_thread)
      for (i=0; i<=4; i++)
      {
                id_thread = omp_get_thread_num( ) ;

                printf(" Iterazione %d del thread %d\n", i, id_thread);
      }


return 0;
}
```

… ancora qualche osservazione sulla combinazione dei costrutti parallel e for

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main( )
{
   int n_threads, id_thread, i;

   omp_set_num_threads(3);
   #pragma omp parallel private(id_thread)
   {
      id_thread = omp_get_thread_num( ) ;
      printf(" Sono %d\n", id_thread);

      #pragma omp for
       for (i=0; i<=4; i++)
      {
                printf(" Iterazione %d del thread %d\n", i, id_thread);
      }
   }

return 0;
}
```

in realtà si tratta di due direttive innestate

… alternativa

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main( )
{
   int n_threads, id_thread, i;

   // COMMENTATO: omp_set_num_threads(3);
   #pragma omp parallel for private(id_thread) num_threads(4)
      for (i=0; i<=4; i++)
      {
            id_thread = omp_get_thread_num( ) ;

            printf(" Iterazione %d del thread %d\n", i, id_thread);
      }


return 0;
}
```

Se non specificato, lo scheduling è quello STATIC. Tuttavia, questo può essere non ottimale per il tipo di problema che devo risolvere! Ad esempio quando differenti iterazioni impiegano un diverso tempo d'esecuzione.

Necessità di scheduling personalizzato !

Guardiamo insieme il seguente codice…
ad ogni iterazione i thread aspettano un numero di secondi pari a l'indice dell'iterazione.

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
#include <unistd.h>

int main( )
{
   int n_threads, i;

   #pragma omp parallel for private(i) schedule(static) num_threads(4)
       for (i=0; i<16; i++)
       {
                //aspetta un numero pari ad i secondi
                sleep(i);
                printf("Il thread %d ha completato iterazione %d.\n", omp_get_thread_num( ) , i);
       }

       printf("Tutti I thread hanno terminato!\n");

return 0;
}
```

provate a far girare il codice e a vedere che succede!

Necessità di scheduling personalizzato: dynamic

E' chiaro che lo scheduling NON è bene organizzato!
Proviamo a migliorare usando la modalità **dynamic**.
In questo modo ciascun thread ha una iterazione: quando questo termina gli verrà assegnata la successiva iterazione.

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main( )
{
   int n_threads, i;

   #pragma omp parallel for private(i) schedule(dynamic) num_threads(4)
      for (i=0; i<16; i++)
      {
                //aspetta i secondi
                sleep(i);
                printf("Il thread %d ha completato iterazione %d.\n", omp_get_thread_num( ) , i);
      }

      printf("Tutti I thread hanno terminato!\n");


return 0;
}
```

Appena un thread finisce, si può occupare di un'altra iterazione!

Necessità di scheduling personalizzato: dynamic

Problemi eventuali nell'uso della modalità dynamic.

La modalità dinamica è migliore quando le iterazioni possono richiedere tempi molto diversi. Tuttavia, esiste un'inevitabile sovraccarico nell'uso della pianificazione dinamica:
dopo ogni iterazione, i thread devono arrestarsi e ricevere un nuovo valore della variabile del ciclo da utilizzare per la successiva iterazione. Questo può provocare un evidente rallentamento.

Cerchiamo di valutare questo rallentamento!

Necessità di scheduling personalizzato: dynamic

Problemi eventuali nell'uso della modalità dynamic.

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#define N 1000

int main( )
{
   int n_threads, i;

   #pragma omp parallel for private(i) schedule(dynamic) num_threads(4)
      for (i=0; i<N; i++)
      {
            printf("Il thread %d ha completato iterazione %d.\n", omp_get_thread_num( ) , i);
      }

      printf("Tutti I thread hanno terminato!\n");


return 0;
}
```

Potrebbe essere più lento della modalità static!

Necessità di scheduling personalizzato: dynamic

Utilizzo del chunk

E' possibile migliorare le prestazioni utilizzando l'opzione **chunk** con la modalità **dynamic**.
In questo modo, ad ogni thread sarà associato un numero prestabilito di iterazioni, quindi quando avrà terminato avrà assegnato un nuovo **chunk**.

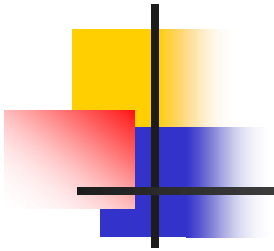
Vediamo nel dettaglio come fare e che succede!

Necessità di scheduling personalizzato: dynamic, chunk

Aumentando la dimensione del chunk lo scheduling tende alla modalità static; mentre diminuendola lo scheduling tende al dynamic.

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#define N 1000
#define CHUNK 10
int main( )
{
   int n_threads, i;

   #pragma omp parallel for private(i) num_thread(4) schedule(dynamic, CHUNK)
       for (i=0; i<N; i++)
       {
                printf("Il thread %d ha completato iterazione %d.\n", omp_get_thread_num( ) , i);
       }

       printf("Tutti I thread hanno terminato!\n");


return 0;
}
```

Necessità di scheduling personalizzato: guided



La modalità **guided** ha una politica di pianificazione molto simile alla modalità **dynamic**, tranne per il fatto che la dimensione del **chunk** cambia durante l'esecuzione del programma.

Inizia con chunk di grandi dimensioni, ma adatta in autonomia la dimensione di chunk, rendendo questo valore più piccolo se il carico di lavoro è sbilanciato.

Vediamo nel dettaglio come fare e che succede!

Necessità di scheduling personalizzato: guided

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#define N 1000

int main( )
{
   int n_threads, i;

   #pragma omp parallel for private(i) num_thread(4) schedule(guided)
       for (i=0; i<N; i++)
       {
                printf("Il thread %d ha completato iterazione %d.\n", omp_get_thread_num( ) , i);
       }

       printf("Tutti I thread hanno terminato!\n");


return 0;
}
```

Provate ad eseguire!

Necessità di scheduling personalizzato: guided

Osservazioni…

Il costrutto FOR divide automaticamente le iterazioni di un ciclo for tra i thread.
A seconda del programma, il comportamento predefinito (modalità STATIC) potrebbe non essere l'ideale.

- Per i cicli for in cui ogni iterazione richiede all'incirca lo stesso tempo, la modalità STATIC funziona meglio, poiché presenta un sovraccarico minimo.
- Per i cicli for in cui le iterazioni possono richiedere quantità di tempo diverse, la modalità DYNAMIC funziona meglio, perché il lavoro verrà suddiviso in modo più uniforme tra i thread, ma ci può essere un ritardo dovuto alla riassegnazione del lavoro.
- Specificare il CHUNK o utilizzare la modalità GUIDED fornisce un compromesso tra le due.

In ogni caso, la scelta migliore dipende dal problema e dalla strategia parallela da implementare

Lezione di lunedì 15

Giornata di Tutoraggio online su teams.

Scrivete per appuntamento – anche di gruppo