Programmazione Avanzata e parallela

Lezione 15

Luca Manzoni

Multithreading con OpenMP

AKA: multithreading made simple(r)

- Cosa è OpenMP
- Utilizzo delle pragma OpenMP
- Sezioni critiche
- Parallelizzare i cicli for
- Non attendere con nowait
- Parallelizzare cicli innestati

OpenMP

Open Multi-Processing

- API per il supporto della programmazione parallela in sistemi multi-processore e memoria condivisa
- Questo il tipo di parallelismo visto utilizzano i thread:
  - Più processori che possono eseguire codice in parallelo
  - Accedendo però alla stessa memoria
- Esistono altre tipologie, e.g., più processori ma memoria non condivisa (in quel caso è comune scambiare messaggi tra i diversi nodi)

OpenMP

Struttura

- Un thread principale fa partire più thread secondari
- Il sistema distribuisce il lavoro tra più thread
- Sono offerti una serie di costrutti utili alla parallelizzazione:
  - Eseguire del codice in parallelo
  - Sincronizzare più thread
  - Parallelizzare i cicli for (in modo quasi immediato)

OpenMP

Utilizzo

- Serve importare il **omp.h** che mette a disposizione una serie di funzioni per ottenere, per esempio, il numero di thread corrente, il numero di thread totali, etc.

- È necessario compilare con l'opzione **-fopenmp**

- All'interno del codice la maggior parte del lavoro è svolto da della pragma ("*The '#pragma' directive is the method specified by the C standard for providing additional information to the compiler, beyond what is conveyed in the language itself.*")…

- …ovvero il supporto di OpenMP è legato al compilatore

OpenMP

#pragma omp parallel

- La prima pragma che è quella che indica che del codice debba essere eseguito in parallelo
- Indica che il blocco di codice successivo (che deve venire racchiuso da graffe se è più di una singola istruzione) deve venire eseguito in parallelo
- Tutte le variabili definite fuori dal blocco sono comuni a tutti i thread (con rischio di race conditions se l'accesso non è regolato)
- Tutte le variabili definite all'interno del blocco sono locali al singolo thread

Pragma omp parallel

Eseguire codice in parallelo

Indichiamo che la parte di codice che segue (racchiusa tra graffe) deve essere eseguita in parallelo su tutti i thread

```
#pragma omp parallel
{
    // codice che verrà eseguito da tutti i thread
}
```

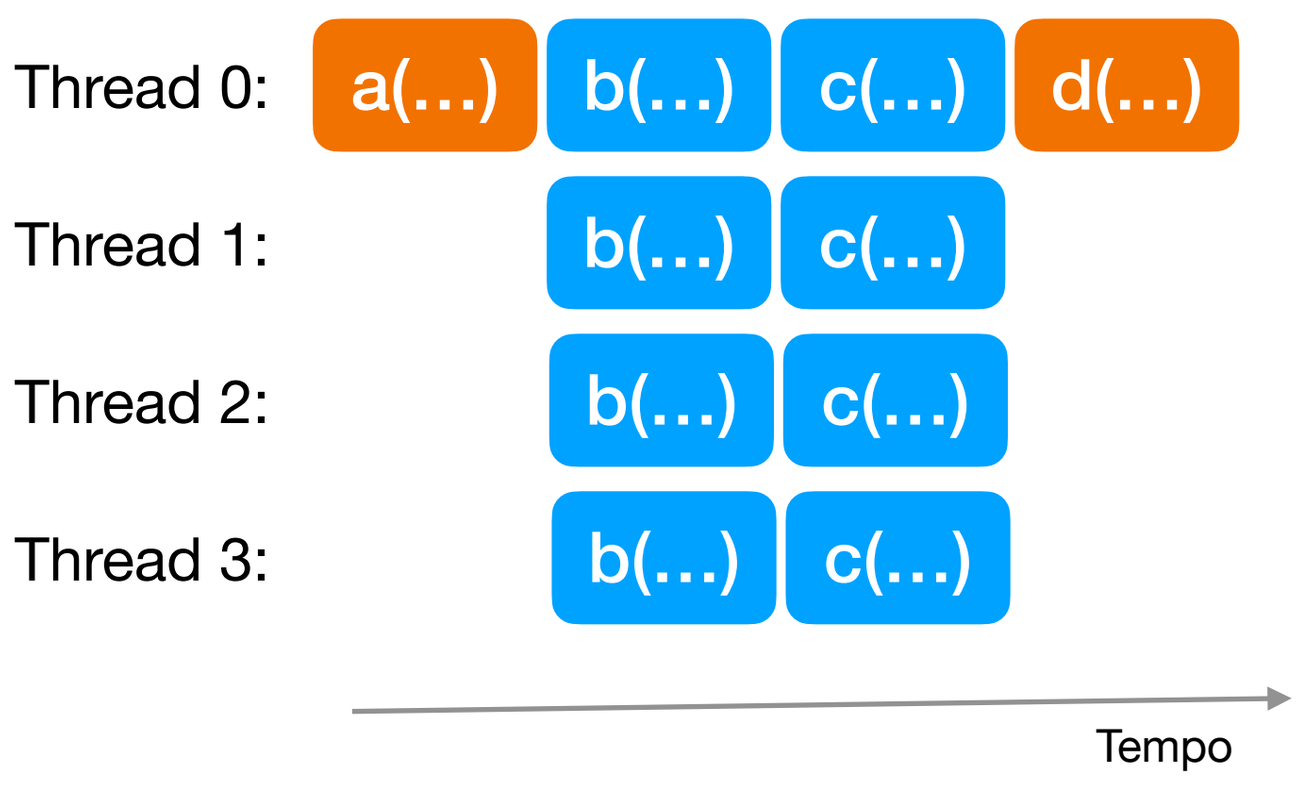
Il codice che viene eseguito è lo stesso per tutti i thread, se vogliamo distinguere quello che viene eseguito dovremo usare, per esempio, un “numero di thread”

Proseguiamo l’esecuzione nel thread principale solo quando la parte parallela viene completa dai diversi thread

Pragma omp parallel

Eseguire codice in parallelo

```
a();
#pragma omp parallel
{
    b();
    c();
}
d();
```



Pragma omp parallel

Eseguire codice in parallelo

```
int x = 2;
#pragma omp parallel
{
    int y = 3;
}
```

Variabile comune a tutti i thread, modificarla può portare a race conditions

Variabile privata per ogni thread (e.g., con 4 thread ne abbiamo 4 copie) può essere modificata senza problemi di race conditions ma non è visibile all'esterno

OpenMP

Far fare cose distinte ai thread

- Se non abbiamo informazioni aggiuntive tutti i thread eseguiranno lo stesso lavoro…
- …ma sarebbe utile che si distinguessero
- Esempio: ogni thread somma un parte dei valori contenuti in un array dividendo quindi il lavoro che ogni thread deve svolgere per il numero di thread.
- Possiamo sfruttare il fatto che ogni thread ha un identificativo da 0 a $n - 1$, dove n è il numero totale di thread

OpenMP

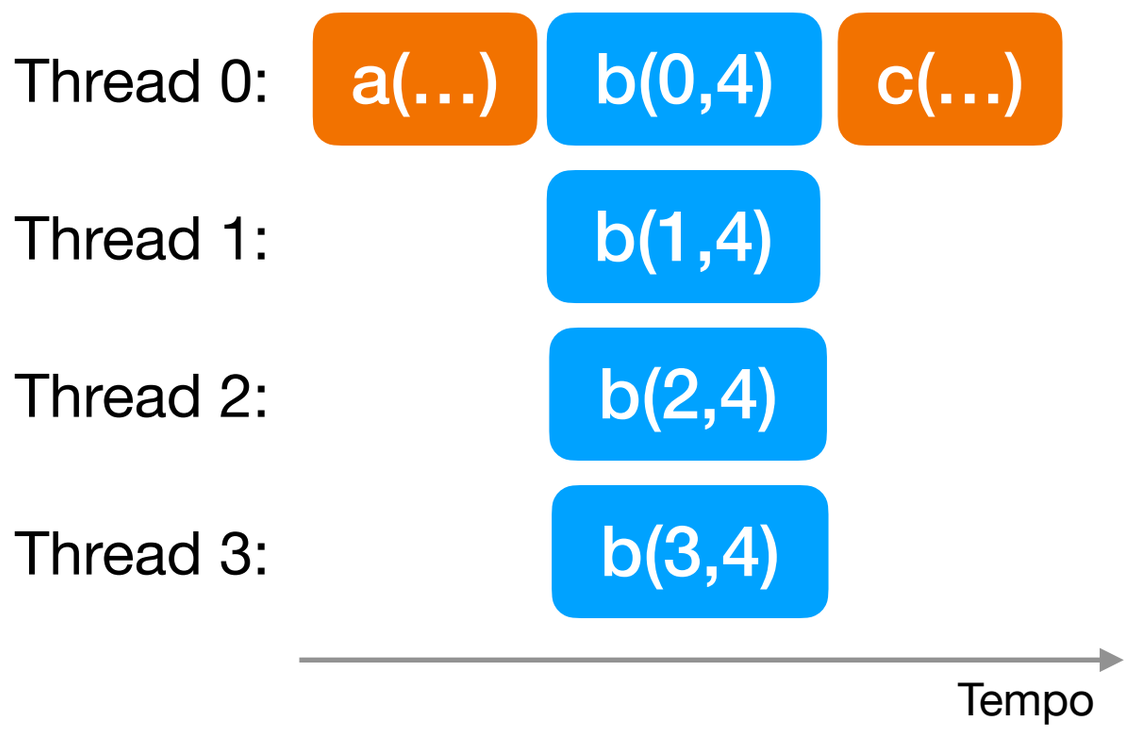
Far fare cose distinte ai thread

- Funzioni da usare fuori dalle sezioni parallele:
  - **omp_get_max_threads()**
    ritorna il numero di thread che verranno usati nelle sezioni parallele
- Funzioni da usare nelle sezioni parallele:
  - **omp_get_num_threads()**
    ritorna il numero di thread che OpenMP sta usando in questa sezione parallel
  - **omp_get_thread_num()**
    ritorna l'identificativo del thread corrente (da 0 a $n-1$)

Pragma omp parallel

Eseguire codice in parallelo

```
a();
#pragma omp parallel
{
    int i = omp_get_thread_num();
    int j = omp_get_num_threads();
    b(i,j);
}
c();
```



OpenMP

Far fare cose distinte ai thread

- Perché ottenere il numero di thread deve essere fatto nelle sezioni e non fuori?

- Possiamo specificare il numero di thread che devono eseguire una sezione parallela

- Esempio:
  #pragma omp parallel num_threads(3)

- Potenzialmente è possibile eseguire anche un solo thread con
  #pragma omp single
  (serve per casi particolari, solitamente all'interno di una sezione parallela)

OpenMP

Sincronizzazione

- Supponiamo di voler scrivere in una variabile comune a tutti i thread, per esempio sommandoci il valore di una loro variabile locale

- Se sommiamo dentro la sezione parallela rischiamo delle race condition…

- …ma fuori dalla sezione parallela le variabili locali non esistono!

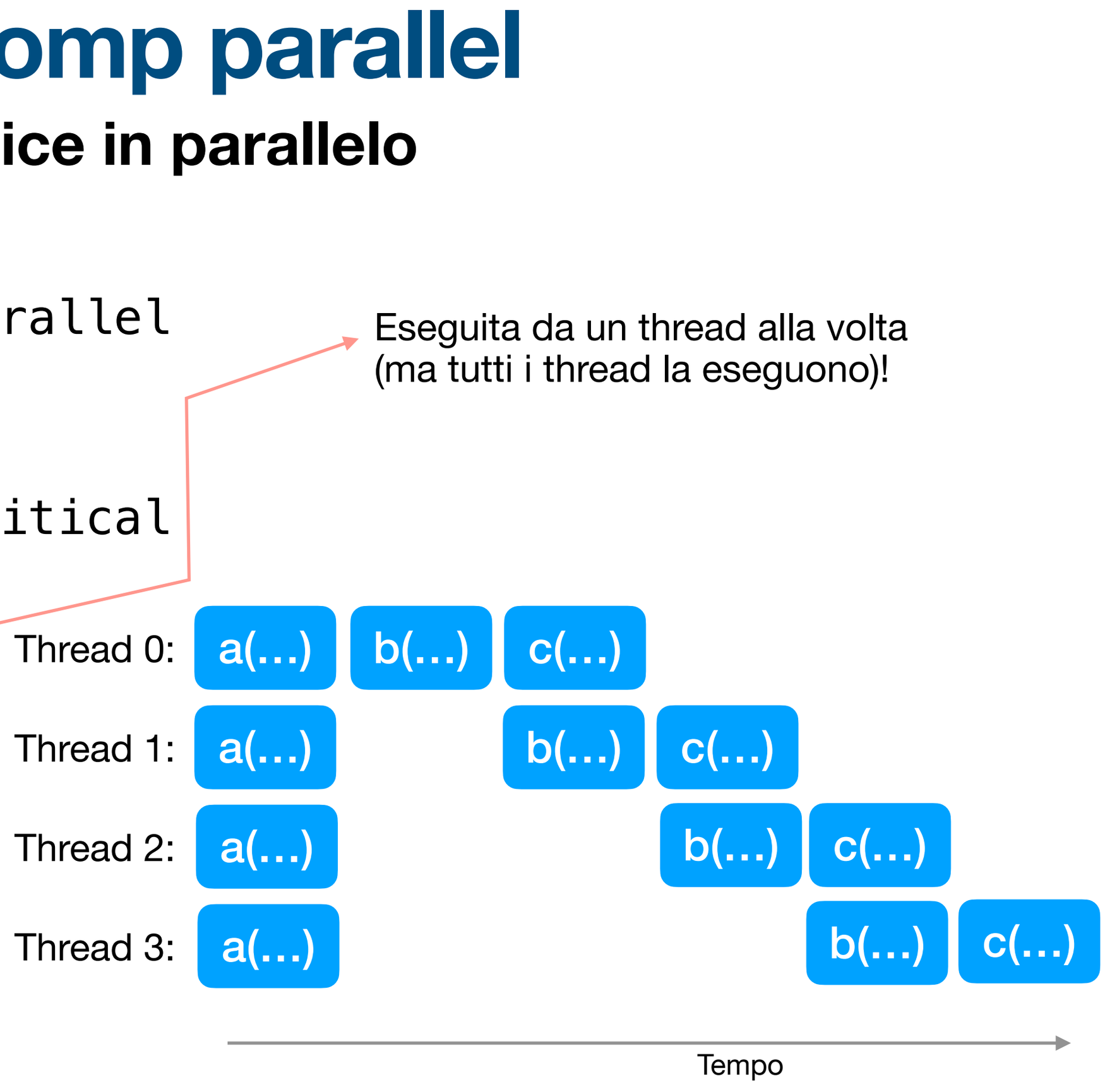
- Possiamo dire che una parte di una sezione parallela è una sezione critica

Pragma omp parallel

Eseguire codice in parallelo

```
#pragma omp parallel
{
    a()
#pragma omp critical
    {
        b();
    }
    c();
}
```

Eseguita da un thread alla volta
(ma tutti i thread la eseguono)!



OpenMP

Sincronizzazione

- Inserire una sezione critica equivale ad aggiungere un lock (specifico per quella sezione):
  - Il lock viene preso all'ingresso della sezione critica…
  - …e rilasciato all'uscita
- Chiaramente le sezioni critiche serializzano l'esecuzione, quindi ha senso usarle solo quando serve!
- Adesso possiamo facilmente scrivere degli algoritmi paralleli che salvano il loro valore alla fine in una memoria condivisa (usando una sezione critica)

OpenMP

Parallel for

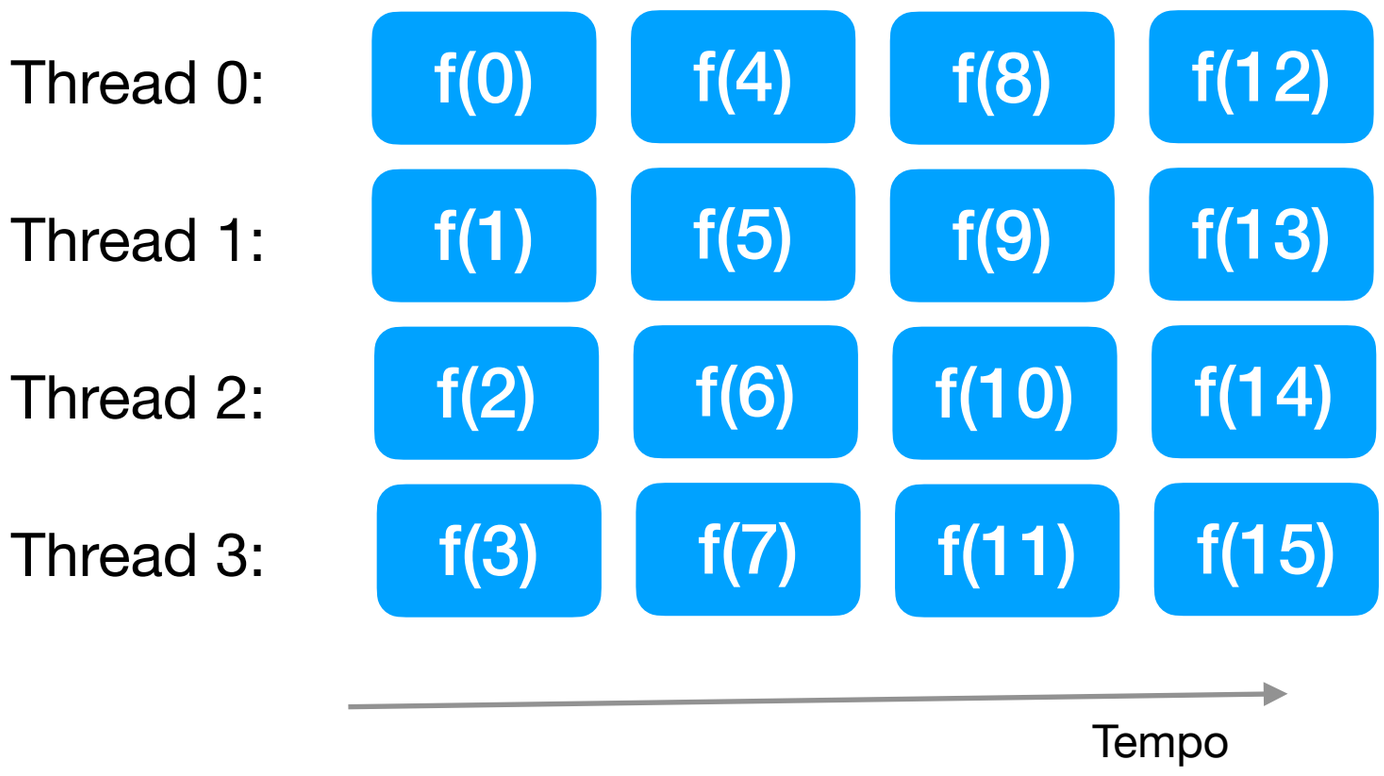
- Un costrutto comune da parallelizzare è il seguente
  - Un ciclo for
    - Ogni iterazione del ciclo opera su variabili differenti, solitamente su indici diversi degli array
- Il metodo standard di parallelizzazione è quello di dare far fare parte del ciclo a ogni thread
- Dato che questo è così comune esiste una pragma apposita:
- **#pragma omp parallel for**

Pragma omp parallel

Eseguire codice in parallelo

Dividi le iterazioni del ciclo for tra i diversi thread

```
#pragma omp parallel for
for (int i = 0; i <16; i++) {
      f(i);
}
```



OpenMP

Parallel for

- La pragma parallel for è la combinazione di due pragma:

- **#pragma omp parallel** (esegue in parallelo)
  #pragma omp for (divide l'esecuzione delle diverse iterazioni del ciclo for tra i diversi thread)

- Se necessario (perché ci sono altre operazioni da far eseguire prima a tutti i thread) è possibile spezzare nelle due pragma

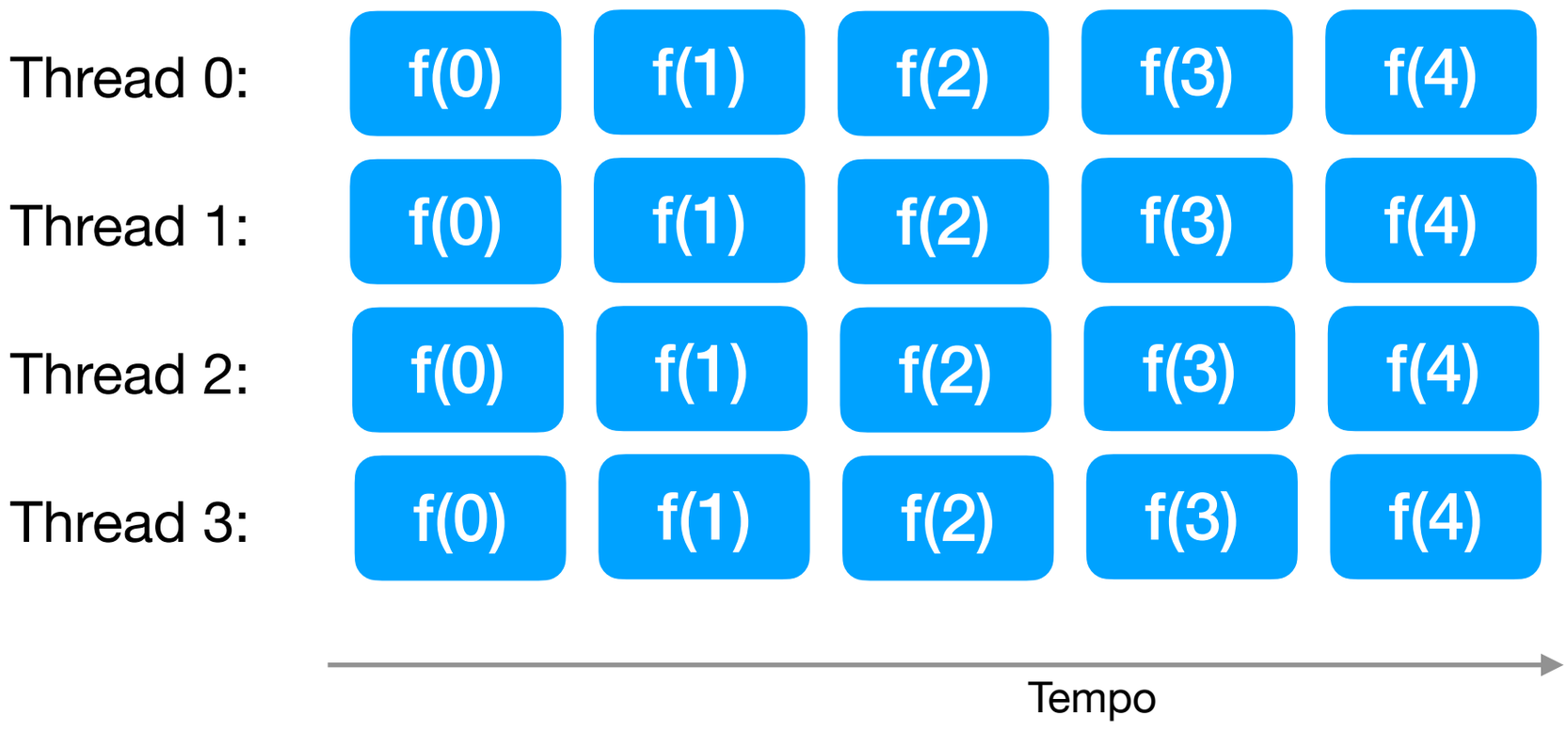
- Mettere un ciclo for dentro una sezione parallela (senza parallel for) non ha la stessa semantica!

Pragma omp parallel

Eseguire codice in parallelo

Le iterazioni del for non sono divise tra i diversi thread ma tutti eseguono il ciclo for!

```
#pragma omp parallel
for (int i = 0; i <16; i++) {
      f(i);
}
```



OpenMP

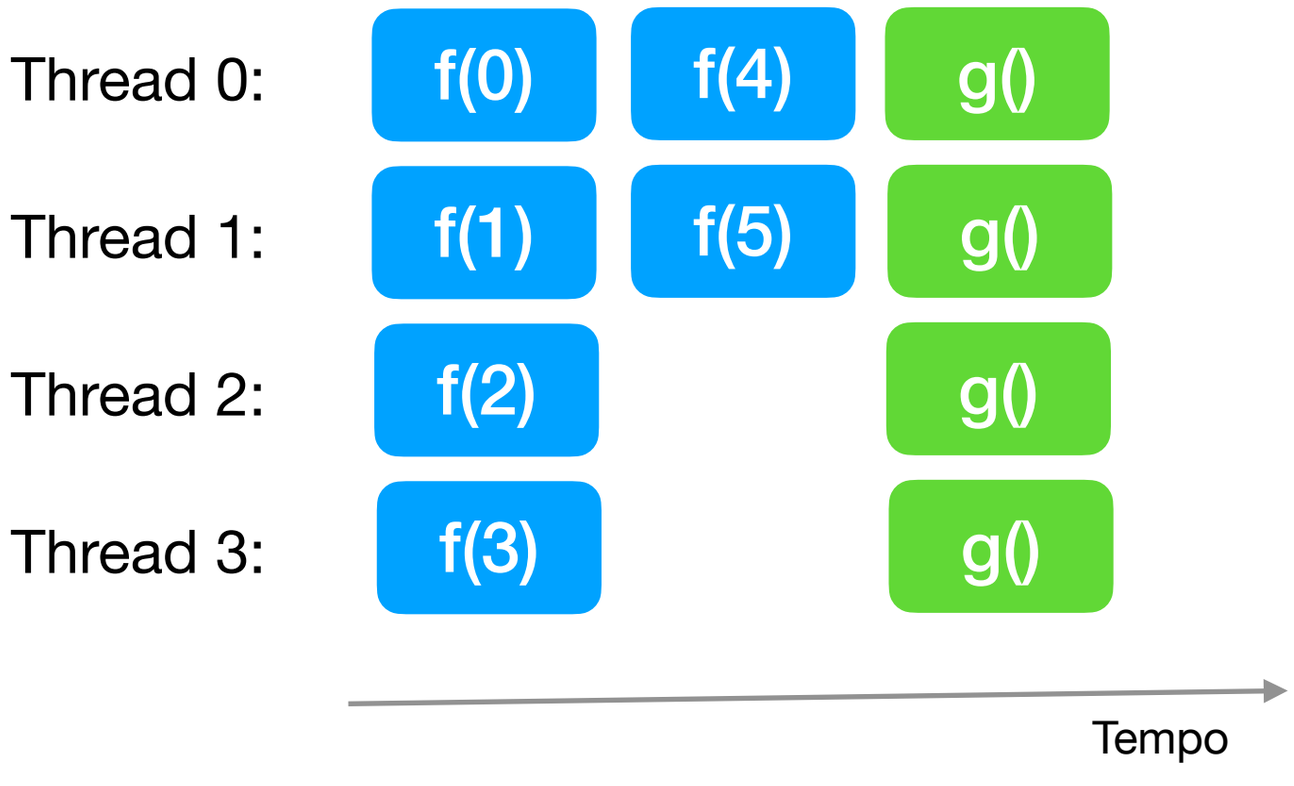
Parallel for e nowait

- Quando usiamo parallel for prima di proseguire (anche se siamo in una sezione parallela) i thread attendono la fine di tutte le iterazioni del ciclo for
- Questo potrebbe portare ad attese non desiderate
- Per dire che non è necessario attendere che tutti i thread terminino è sufficiente aggiungere **nowait** alla pragma for:
- **#pragma omp for nowait**

Pragma omp parallel

Eseguire codice in parallelo

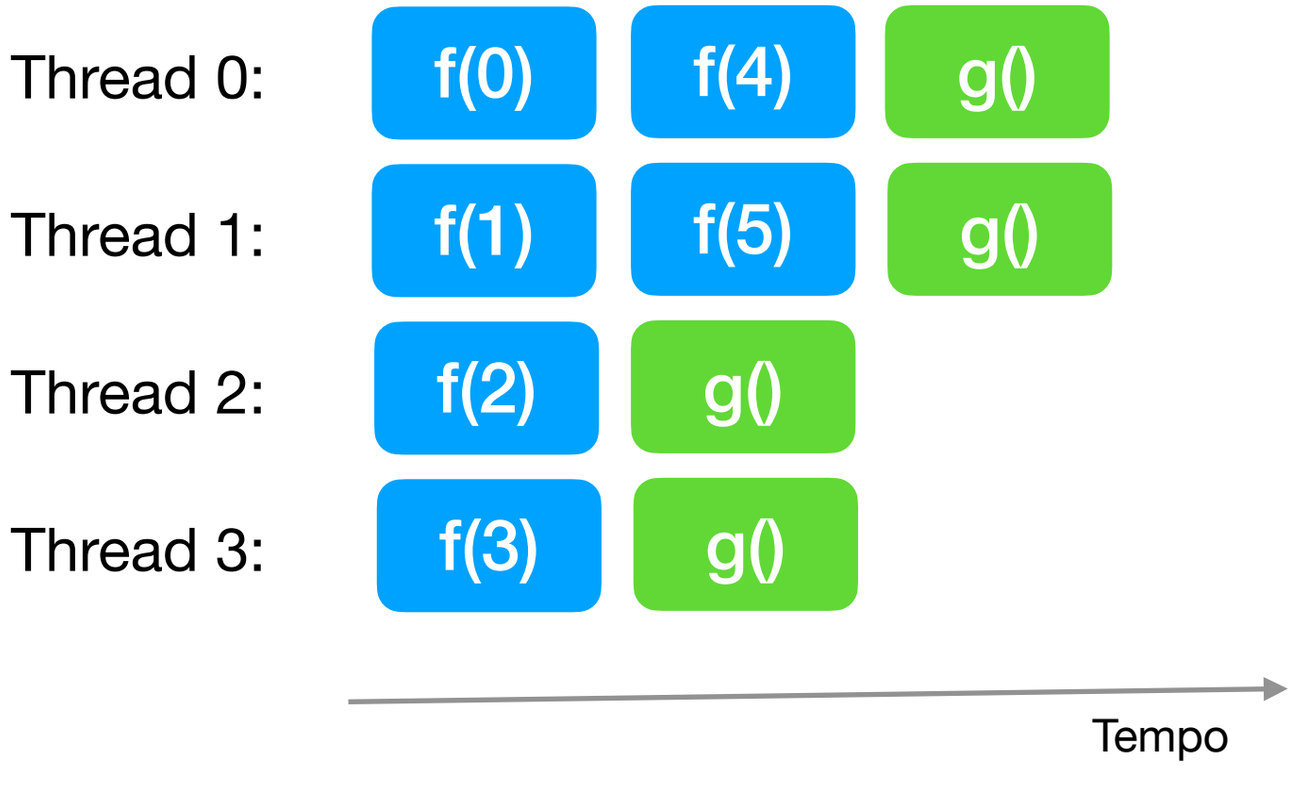
```
#pragma omp parallel
{
    #pragma omp for
    for (int i = 0; i < 6; i++) {
        f(i);
    }
    g();
}
```



Pragma omp parallel

Eseguire codice in parallelo

```
#pragma omp parallel
{
    #pragma omp for nowait
    for (int i = 0; i < 6; i++) {
        f(i);
    }
    g();
}
```



OpenMP

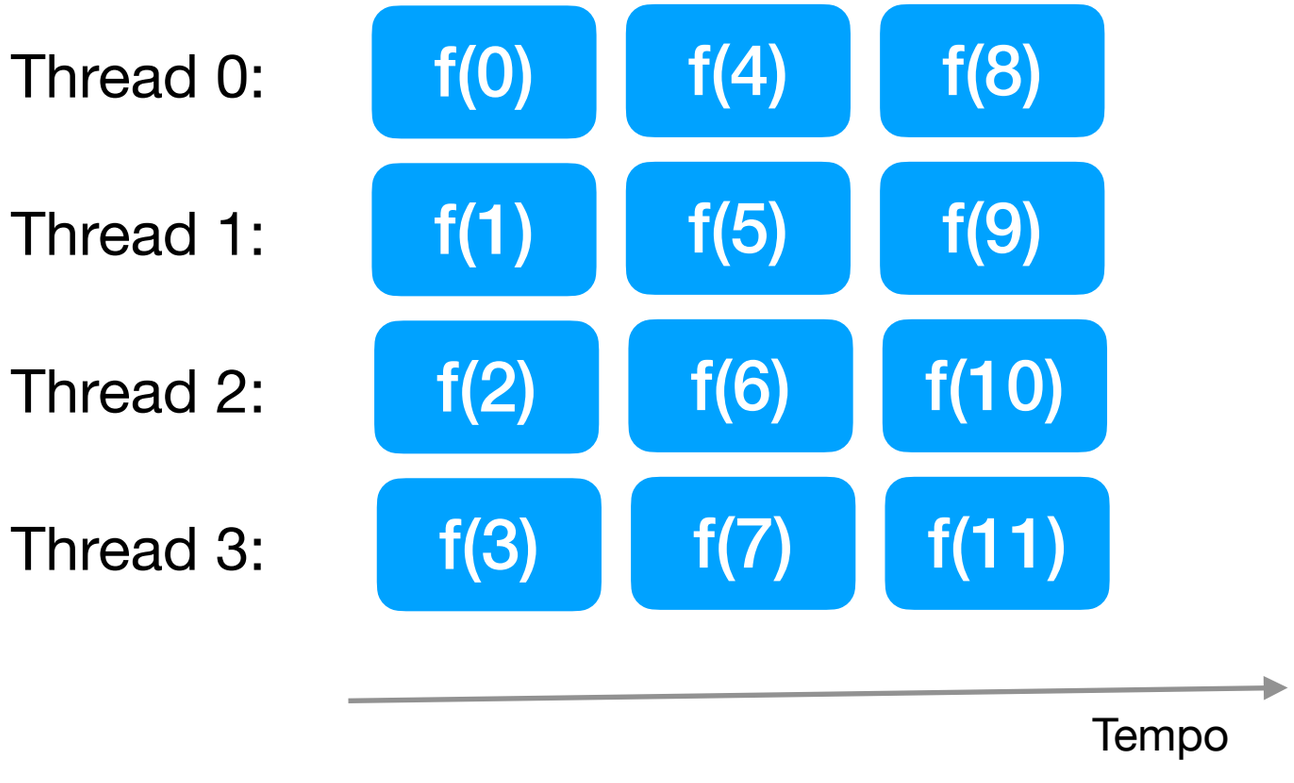
Schedule

- Possiamo decidere come sono spezzati tra i thread i cicli for
- Di default si assegna la prima iterazione al primo thread, la seconda al secondo, e via così ricominciando da capo
- Questo è lo scheduler statico con dimensione 1
- Possiamo anche decidere di cambiare la dimensione a k, in quel caso daremo k iterazioni al primo thread, le k successive al secondo, etc.
- Utile nel caso possa servire che cicli consecutivi siano “vicini” (e.g., questioni di località di memoria)

Pragma omp parallel

Schedule

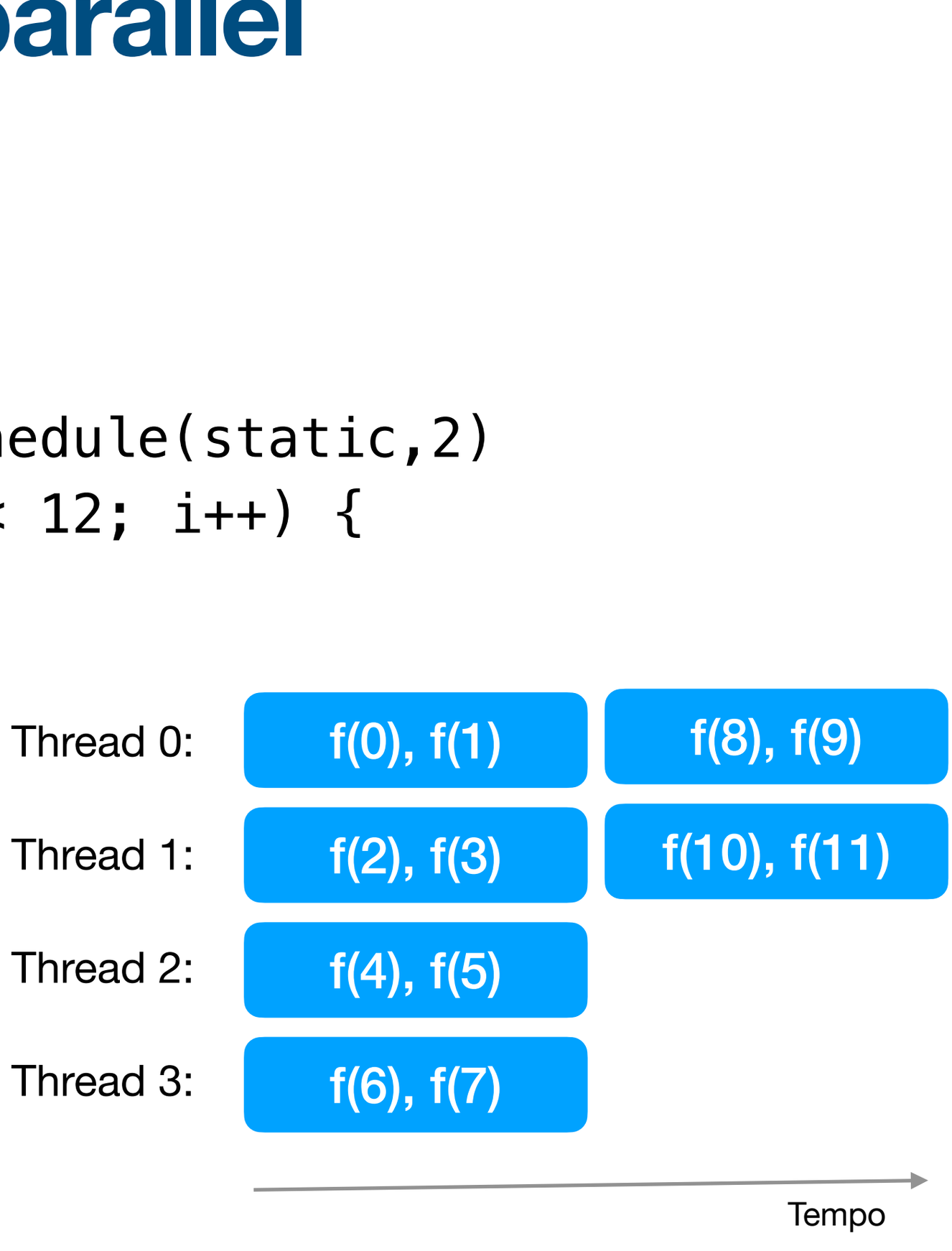
```
#pragma omp parallel
{
    #pragma omp for schedule(static,1)
    for (int i = 0; i < 12; i++) {
        f(i);
    }
    g();
}
```



Pragma omp parallel

Schedule

```
#pragma omp parallel
{
    #pragma omp for schedule(static,2)
    for (int i = 0; i < 12; i++) {
        f(i);
    }
    g();
}
```

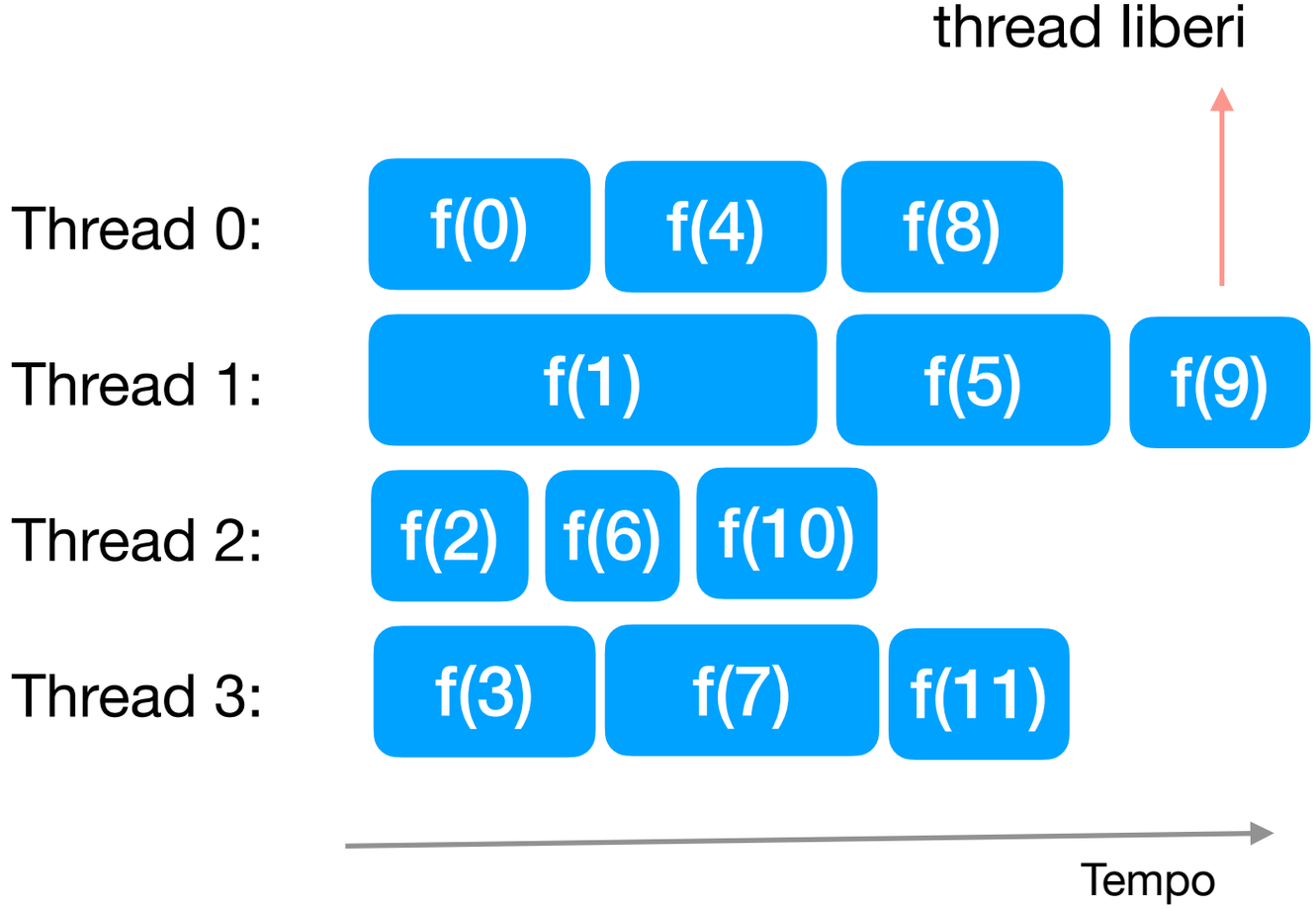


Pragma omp parallel

Schedule

```
#pragma omp parallel
{
    #pragma omp for
   for (int i = 0; i < 12; i++) {
        f(i);
    }
    g();
}
```

Non essendo iterazioni della stessa lunghezza attendiamo che un thread completi quando ci sono thread liberi



OpenMP

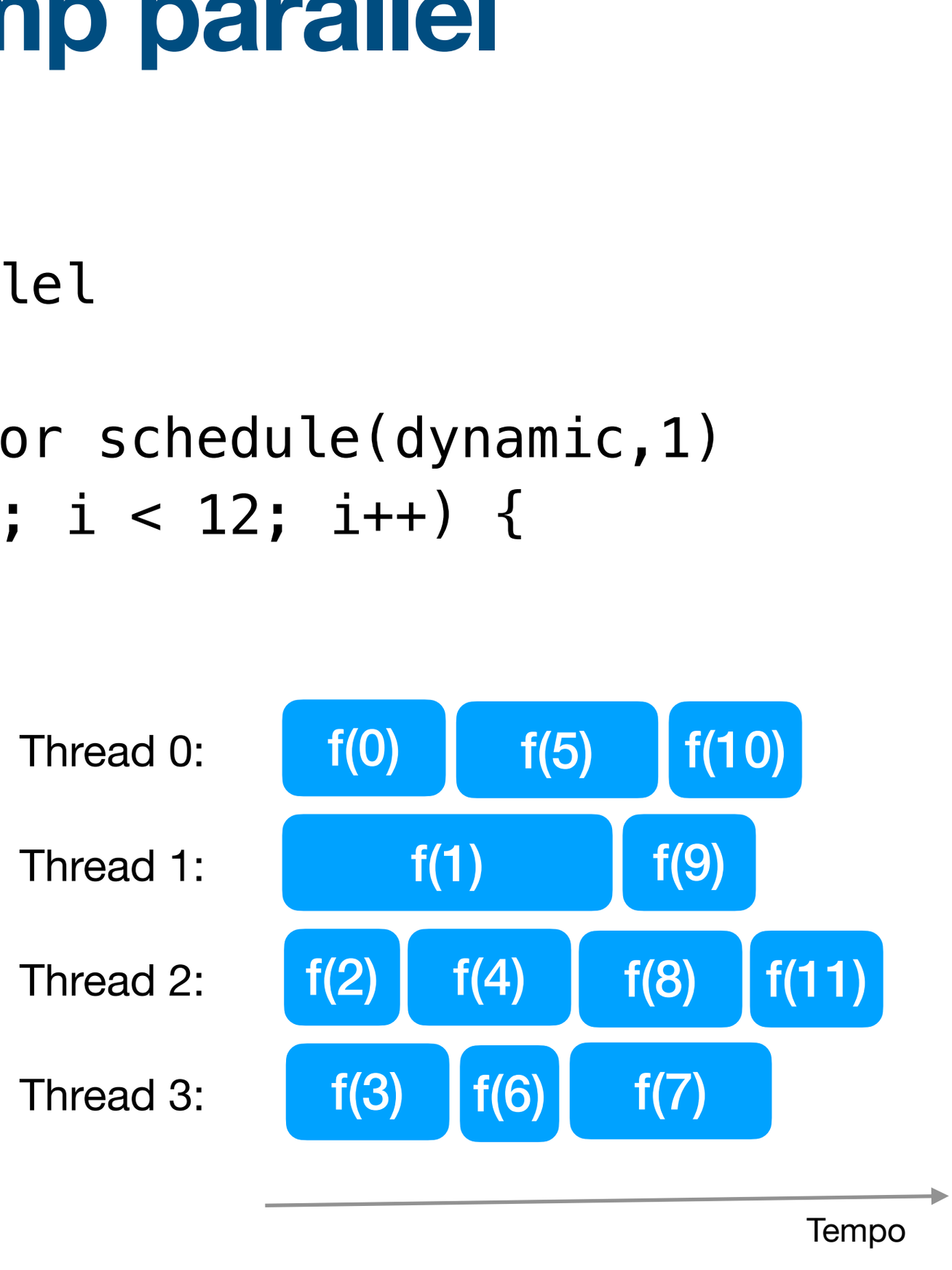
Schedule dinamico

- Nel caso non tutte le iterazioni abbiamo la stessa lunghezza lo schedule statico potrebbe non essere ottimale
- Possiamo usare uno schedule dinamico:
  - Dopo aver completato un blocco di iterazioni il thread verifica quale iterazione del ciclo for non sta venendo eseguita da nessuno e la prende
- Questa operazione di scheduling è costosa, quindi va fatta solo se c’è questo sbilanciamento…
- …ed è utile aumentare la dimensione dei blocchi per trovare un trade-off tra quanto ribilanciare e quanto di frequente eseguire lo scheduling

Pragma omp parallel

Schedule

```
#pragma omp parallel
{
     #pragma omp for schedule(dynamic,1)
    for (int i = 0; i < 12; i++) {
            f(i);
    }
    g();
}
```



OpenMP

Scheduling e nesting

- Rimangono aperte alcune domande:
  - Come parallelizzare loop innestati?
  - Possiamo usare del parallelismo “task-based” (avere una lista di compiti da “mettere in coda” e i thread prendono il primo lavoro ancora da fare dalla coda)?
  - Spesso accumuliamo il risultato di una operazione in una variabile: possiamo automatizzare la cosa?
  - Istruzioni atomiche e il modello di memoria di OpenMP