Programmazione Avanzata e parallela

Lezione 16

Luca Manzoni

Multithreading con OpenMP

Task, riduzioni, cicli innestati

- Riduzione per i cicli
- Come gestire cicli innestati
- Parallelismo per task

Riduzione per cicli for

Accumulare in una variabile condivisa

- Un pattern comune sui cicli for è quello di accumulare un valore in una variabile condivisa:
  - Somma di tutti gli elementi del vettore
  - Trovare se almeno uno/tutti degli elementi del vettore rispettano una condizione
- Per fare questo solitamente si usa una variabile che viene aggiornata ad ogni iterazione del ciclo:
  - e.g., `sum += a[i]`

Riduzione per cicli for

Accumulare in una variabile condivisa

- In casi come questo usare #pragma omp parallel for non va bene: c’è una variabile che è acceduta in scrittura da tutti i thread
- Soluzione “a mano”
  - Creare una variabile locale per ogni thread che tiene la somma parziale
  - Alla fine del ciclo for calcolare la somma totale aggiungendo le somme parziali ad una variabile condivisa (proteggendola in una sezione critica)

Riduzione per cicli for

Accumulare in una variabile condivisa

- Questa pattern di soluzione è comune (e non solo per la somma) ed è quindi gestibile in modo automatico da OpenMP

- Per effettuare una riduzione (ovvero per combinare i risultati intermedi) su una specifica variabile usiamo
#pragma omp for reduction(op : var)
dove:

- “op” è l’operazione, una tra: +, -, *, &, |, ^, &&, ||

- “var” è la variabile (usata all’interno del loop) su cui effettuare la riduzione

Riduzione per cicli for

Accumulare in una variabile condivisa

- Stiamo dicendo che calcoleremo (per ogni iterazione del ciclo) un valore cha verrà combinato con var tramite l'operazione op

- Esempio:

```
#pragma omp for reduction(+: sum)
for (int i = 0; i < n; i++) {
    sum += v[i];
}
```

- OpenMP creerà in automatico le "somme parziali" (locali a ogni thread) ed effettuerà la somma finale in automatico (una copia "locale" di sum verrà creata in automatico)

Riduzione per cicli for

Accumulare in una variabile condivisa

- Ci sono alcune limitazioni perché questo funzioni.
- Nel corpo del metodo solo le seguenti forme di costrutti riguardanti var sono supportate:
  - **var = var op expr** (expr non deve dipendere da var)
  - **var = expr op var**
  - **var op= expr**
  - **var++**; **var--**; **++var**; **--var**; (solo per operatori + e -)

Cicli innestati

Come parallelizzarli

- Solitamente è sufficiente inserire
  #pragma omp parallel for
  solo per il ciclo più esterno

- In questo caso le iterazioni del ciclo esterno sono divise tra più thread, questo non vale per le iterazioni interne

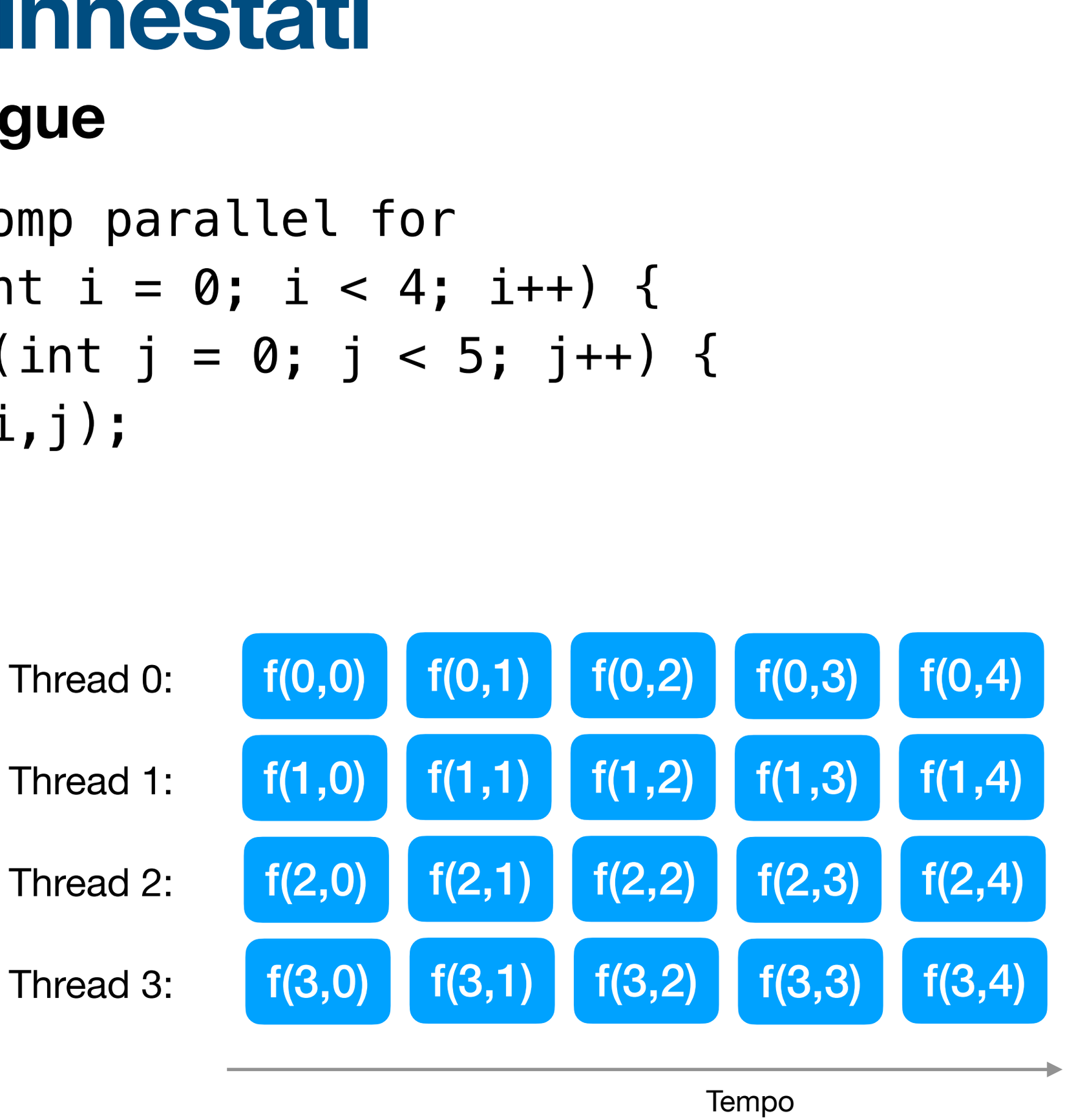
- Se abbiamo 4 thread e $i = 0,\ldots, n-1$ (ciclo esterno) mentre $j = 0,\ldots, m-1$ (ciclo interno), parallelizzando il ciclo esterno otteniamo la seguente divisione degli indici:

- Thread 0: $(0,0), (0,1), \ldots, (0,m)$ seguito da $(4,0), (4,1), \ldots, (4,m)$, etc

Cicli innestati

Chi esegue

```
#pragma omp parallel for
  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 5; j++) {
      f(i,j);
    }
  }
}
```



Cicli innestati

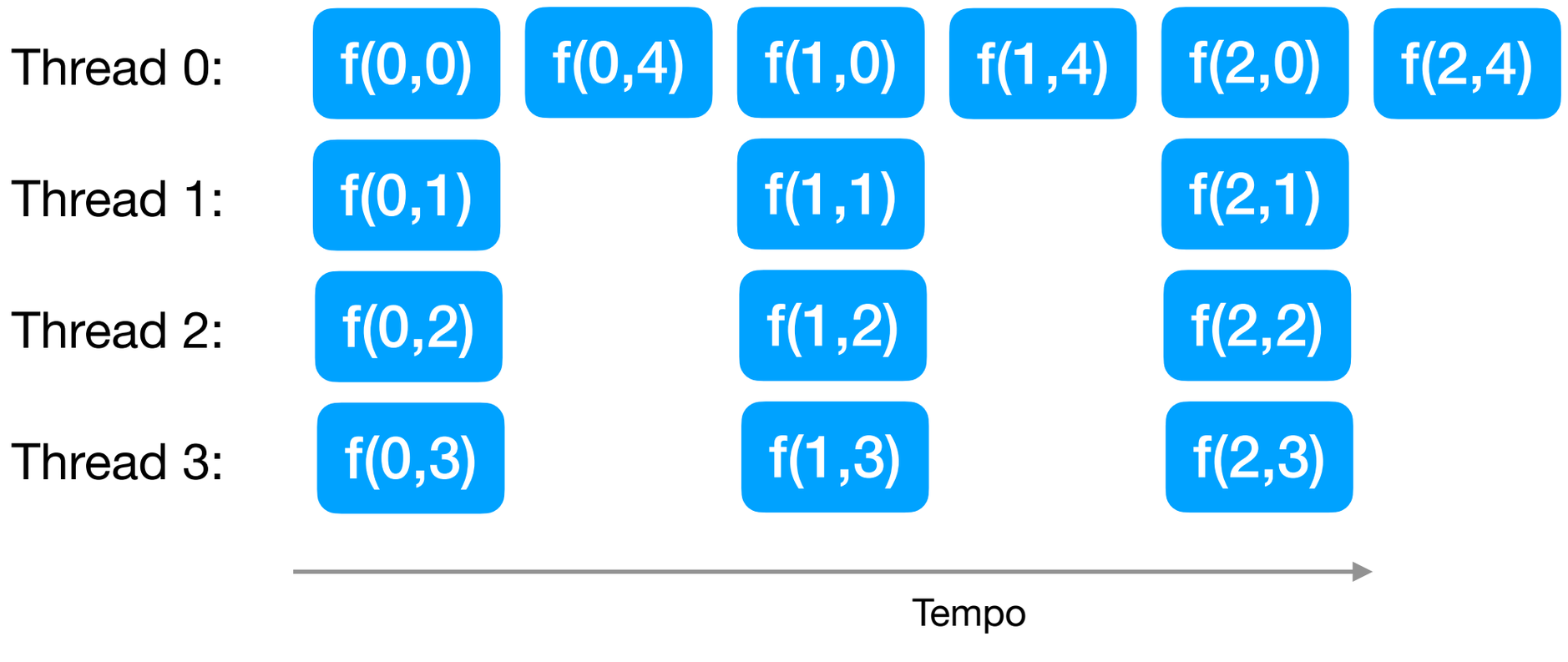
Come parallelizzarli

- Se il ciclo esterno è molto corto potremmo parallelizzare il ciclo interno…
- …ma ogni iterazione del ciclo esterno ha un momento in cui tutti i thread si aspettano
- Perchè?
  - Le iterazioni del ciclo esterno avvengono in modo sequenziale
  - Quindi prima di proseguire dobbiamo aspettare la del ciclo interno (che avviene in parallelo)

Cicli innestati

Chi esegue

```
for (int i = 0; i < 4; i++) {
#pragma omp parallel for
  for (int j = 0; j < 5; j++) {
    f(i,j);
  }
}
```



Cicli innestati

Cose da non fare

- E se aggiungessimo un **#pragma omp for** a ogni ciclo?
- Non serve assolutamente a nulla, è equivalente all'aggiunta solo al ciclo più esterno
- Idea intuitiva: quando si incontra il secondo "parallel for" c'è un solo thread che sta eseguendo quella specifica iterazione, quindi non può dividere il lavoro con altri thread
- Mettere tutti i cicli for all'interno di una sezione parallela ma usare **#pragma omp for** solo per i cicli interni
- Questo provoca soluzioni sbagliate (ciclo for eseguito tante volte quanti sono i thread)

Cicli innestati

Come parallelizzarli

- La soluzione corretta è “collassare” due cicli innestati di lunghezza n e m in un solo ciclo di lunghezza $n \times m$
- Servirà ricavarci le variabili indice dei cicli originali usano operazioni di modulo e divisione senza resto
- L’operazione di collasso di più cicli innestati può essere fatta in automatico con l’opzione collapse di **#pragma omp for**:
- **#pragma omp for collapse(2)**
  “collassa” i due cicli più esterni che seguono

Task

Come parallelizzarli

- Supponiamo di avere funzioni (anche molto differenti) che possono eseguire in parallelo
- Possiamo pensare di avere un “pool” di thread pronti a eseguire compiti (i.e., siamo dentro una sezione parallela)
- Abbiamo un unico thread che dispensa i compiti (i.e., siamo dentro una sezione “single”)…
- …e questi vengono eseguirti da altri thread
- Questo è fattibile con **#pragma omp task**

Task

```
#pragma omp parallel
#pragma omp single
{
  f(2);
#pragma omp task
  g(3);
#pragma omp task
  f(12);
  g(24);
  f(3);
#pragma omp task
  g(5);
  g(8);
}
```