TPA 2023/24 – Heat

Questo è il testo dell'esame del corso "Tecniche di Programmazione Avanzata" del Dipartimento di Ingegneria Industriale, anno accademico 2023/2024.

Istruzioni

L'esercizio consiste nella progettazione e nell'implementazione di un software C++ e OpenMP per la simulazione della dissipazione di calore in una superficie 2D. Il software deve essere scritto mediante il paradigma di programmazione ad oggetti, quindi utilizzando ad esempio (ove possibile e sensato):

- classi
- incapsulamento
- ereditarietà
- polimorfismo
- passaggio per riferimento
- passaggio per riferimento costante
- puntatori
- metodi const
- operatori
- overload
- valori di default
- template

Nessun codice di partenza è fornito. Lo studente deve occuparsi sia della progettazione che dell'implementazione. Si consiglia di consegnare anche il progetto cartaceo del software, soprattutto se l'implementazione dovesse essere incompleta (ad esempio per mancanza di tempo). Il progetto sarà tenuto in conto nella valutazione finale.

L'implementazione deve includere anche una serie di test per verificare il corretto funzionamento del codice scritto.

Descrizione

Dovete scrivere un software parallelo per la simulazione della dissipazione di calore in una superficie 2D. La simulazione lavora su una superficie (matrice) di NxN valori in double precision floating point. Ogni valore rappresenta la temperatura $T_{i,j}$. Per ogni passo della simulazione (t) ogni temperatura ($T_{i,j}$) viene attualizzata secondo la seguente formula:

$$T_{i,j}^{t+1} = T_{i,j}^{t} + \alpha \cdot \Delta t \cdot (T_{i+1,j}^{t} + T_{i,j+1}^{t} + T_{i-1,j}^{t} + T_{i,j-1}^{t} - 4 \cdot T_{i,j}^{t})$$

Dove alpha è il coefficiente di diffusione e delta t è l'ampiezza dell'intervallo di tempo considerato per la simulazione.

Per semplificare la scrittura del codice considerate i valori di temperatura ai confini della superficie costanti a zero ($T_{i,0} = T_{0,j} = 0$) senza doverli attualizzare durante la simulazione.

La simulazione che aggiorna i valori della temperatura deve continuare finché si sia raggiunto uno stato sufficientemente stabile, cioè finchè la somma del cambiamento di temperatura nell'intera superficie da uno step al successivo non sia minore di un epsilon prestabilito (la differenza fra la somma delle $T_{i,j}$ allo step t e la somma delle $T_{i,j}$ allo step t-1 sia minore di epsilon).

Implementazione

Per questo esercizio considerate le seguenti costanti:

$$N = 32, \alpha = 0.5, \Delta t = 0.1, \epsilon = 0.001$$

All'inizio della simulazione **inizializzate tutti i valori $T_{i,j} = 0$** ma si deve mantenere costante **$T_{5,5} = 5$ e $T_{20,20} = 3$ per tutta la simulazione.**

Quando la simulazione termina stampate (su file output.txt) la matrice risultato riga per riga, usando %3.2f come formato di scrittura, e uno spazio fra le colonne. Aggiungete un newline ('\n') alla fine di ogni riga. Avete a disposizione sia un file **input_heat.txt** con la matrice di input giá impostata, se preferite leggere da file.

Per scrivere il codice creare una classe che abbia i seguenti metodi:

costruttore: inizializza la matrice input secondo le indicazioni

update: esegue uno step della simulazione

sumT: somma le temperature di tutti gli elementi della matrice

Creare inoltre un programma di test (main) che esegua le operazioni richieste e stampi la matrice finale in un file output.txt al termine della simulazione.

Ovviamente si devono parallelizzare in OpenMP tutte le operazioni parallelizzabili, garantendo la correttezza del risultato. Il numero di threads da instanziare é indifferente per questo esercizio (non si devono considerare le performance).

Suggerimenti:

-Scrivere il codice sequenziale e poi passare al parallelo, parallelizzando un'operazione alla volta.

-State attenti alla parallelizzazione e al fatto che per attualizzare $T_{i,j}$ sono necessari i valori delle temperature dei vicini non attualizzati. Potete scegliere a che livello parallelizzare e come fare in modo di evitare la concorrenza di dati senza troppe complicazioni.

-Per controllare la correttezza del codice provate con una matrice semplificata (tutti 0 e un elemento solo con un valore diverso e stampate l'uscita ogni pochi step, così da controllare che la temperatura del valore diverso da zero si diluisca fra i vicini). Avete a disposizione uno script python ("**display_mxm.py**") che vi mostra la bitmap della matrice output.txt per visualizzare meglio il risultato.

Semplificazioni (con riduzione del voto):

-Chi avesse problemi con il controllo del raggiungimento dello stato stabile può limitare la simulazione ad un numero fissato di step (modificabile nel codice), e provare con numeri sufficientemente alti.

-Chi avesse problemi con il mantenere costanti $T_{5,5} = 5$ e $T_{20,20} = 3$ può trattarli come punti normali da attualizzare (ma sempre inizializzati a 5 e 3, rispettivamente)

-Chi avesse problemi con la creazione della classe e dei metodi può scrivere tutte le funzioni nel main.

-Chi avesse problemi con la parallelizzazione può lasciare il codice sequenziale.

Complicazioni (con aumento del voto):

-Dare la possibilità di scegliere le variabili N, alpha, epsilon da riga di comando.

-Considerare i valori ai bordi della superficie non costanti, ma uguali ai valori immediatamente più interni ($T_{i,0} = T_{i,1}$), e quindi da attualizzare ad ogni step.