Corso di High Performance Computing

Esercitazione OpenMP del 30/3/2017

Moreno Marzolla

Ultimo aggiornamento: 2017/03/29

Per svolgere l'esercitazione è possibile collegarsi al server `disi-hpc.csr.unibo.it` tramite ssh, usando come *username* il proprio indirizzo mail istituzionale completo, e come password la propria password istituzionale (cioè quella usare per accedere alla casella di posta o ad AlmaEsami). Sulla macchina è installato il compilatore `gcc` e alcuni editor di testo per console: `vim`, `pico`, `joe`, `ne` e `emacs`. Per chi non è pratico suggerisco `pico`, che è semplice da usare e richiede poche risorse. Chi ha un portatile con Linux può lavorare localmente, se ha installato il compilatore.

Per scaricare l'archivio con i sorgenti di questa esercitazione è possibile usare i comandi:

```
wget http://www.moreno.marzolla.name/teaching/HPC/ex2-openmp.zip
unzip ex2-openmp.zip
cd ex2-openmp/
```

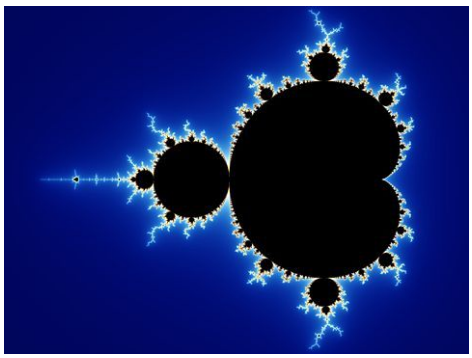
1. Prodotto matrice-matrice

Il file `omp-matmul.c` contiene l'implementazione seriale dell'algoritmo cache-efficient per calcolare il prodotto matrice-matrice che abbiamo visto in una delle precedenti lezioni.

1. Analizzare il codice e parallelizzarlo ove possibile usando le direttive OpenMP appropriate.
2. Valutare sperimentalmente lo speedup della versione parallela, fissata una dimensione delle matrici che consenta di osservare tempi significativi.

2. Area dell'insieme di Mandelbrot

L'insieme di Mandelbrot corrisponde alla parte nera nella figura seguente.



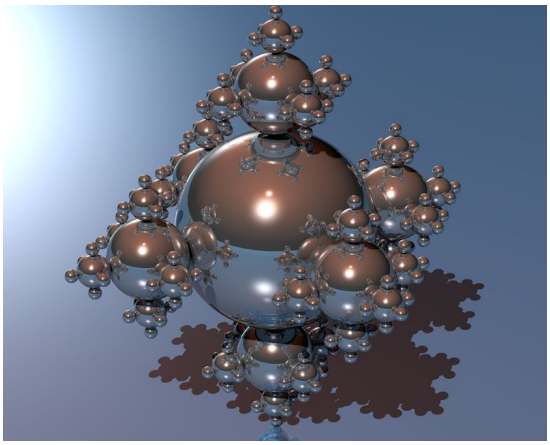
Nel file `omp-mandelbrot-area.c` viene fornita la versione seriale di un programma che stima l'area dell'insieme di Mandelbrot. Il surgente include le istruzioni per la compilazione e l'esecuzione.

1. Compilare ed eseguire il programma per verificarne il funzionamento.
2. Modificare il programma per sfruttare il parallelismo OpenMP sfruttando le direttive e le clausole che si ritengono necessarie.
3. Studiare sperimentalmente se e come la scelta del tipo di scheduling (*static* o *dynamic*) e della dimensione dei blocchi (*chunksize*) influisca sul tempo di esecuzione del programma. Non è ovviamente possibile esaminare tutti i valori di *chunksize*, quindi si consiglia di usare

un insieme limitato di valori per avere una idea grossolana del comportamento del programma. Suggerimento: se il programma impiega troppo tempo (o troppo poco) consiglio di modificare il valore del simbolo `NPOINTS` nel sorgente.

3. Ray tracing

Nella directory `c-ray/` è presente il codice sorgente di un semplice programma di ray-tracing (scritto da John Tsiombikas e rilasciato con licenza GPLv2). Il programma è contenuto in un unico sorgente C che include le istruzioni per la compilazione e l'esecuzione; è anche presente un file README con maggiori informazioni, per chi fosse interessato.
Assieme al programma sono forniti due file di input: `sphfract.small.in` e `sphfract.big.in`. Il primo contiene una scena semplice che può essere elaborata velocemente; il secondo contiene la stessa scema con maggiori dettagli, e richiede un tempo di calcolo maggiore. I file producono entrambi una figura come questa, eventualmente con un livello di dettaglio maggiore.



1. Compilare ed eseguire il programma (si consiglia di usare il file `sphfract.small.in` come input), seguendo le istruzioni presenti nel file sorgente.
2. Modificare il programma per sfruttare il parallelismo fornito da OpenMP. Suggerimento: concentrarsi sulla funzione `render()`.
3. Calcolare lo *speedup* e l'efficienza della versione parallela, utilizzando un numero di thread OpenMP che varia ad esempio da 1 a 4 oppure da 1 a 8, usando se possibile `sphfract.big.in` come input; se i test richiedono troppo tempo usare l'altro file. Per misurare i tempi è necessario ripetere ciascun test più volte (ad esempio 5), calcolando poi il tempo medio di esecuzione come illustrato a lezione.

Chi vuole visualizzare l'immagine risultante può usare il comando `display` (in tal caso è necessario collegarsi al server usando Cygwin/X); in alternativa è possibile copiare il file sul PC locale e visualizzarlo lì.

4. Algoritmo di Bellman-Ford

In questo esercizio viene richiesta l'implementazione e la parallelizzazione dell'algoritmo di Bellman-Ford per il calcolo dei cammini minimi da singola sorgente (*Single-Source Shortest Paths*). Dato un grafo orientato pesato con n nodi e m archi con pesi non negativi, l'algoritmo calcola le distanze $d[j]$ di ciascun nodo j, $0 \leq j < n$ da un nodo sorgente s (la distanza del nodo sorgente da se stesso è zero, quindi $d[s] = 0$).
L'algoritmo effettua n passi di "rilassamento", in cui ad ogni passo tenta di migliorare la stima delle distanza di ciascun nodo dalla sorgente. Un passo di rilassamento consiste nell'esaminare ciascun arco (i, j) del grafo: detto $w(i, j)$ il peso dell'arco (i, j), se $d[i] + w(i, j) < d[j]$ significa che il cammino dalla sorgente al nodo i unito all'arco (i, j) permette di raggiungere il nodo j con un costo

inferiore alla precedente stima $d[j]$. In tal caso si aggiorna la stima come $d[j] \leftarrow d[i] + w(i, j)$.
Allo scopo di ridurre (ma non evitare!) *race conditions*, l'algoritmo seguente fa uso di due array per le distanze: $d[]$ indica la stima "corrente" delle distanze minime, mentre *dnew*[] indica la stima "migliorata" dopo aver applicato un passo di rilassamento. Dopo ogni fase di rilassamento è necessario copiare il contenuto di *dnew*[] in $d[]$ (non bisogna scambiarli tra loro!).

```
BellmanFord(grafo G=(V, E, w), int s, double d)
    double dnew[0..n - 1];
    // Inizializza tutte le distanze a +∞
    for i ← 0 to n – 1 do
        d[i] ← dnew[i] ← +∞
    endfor
    d[s] ← dnew[s] ← 0;
    // Esegui n passi di rilassamento
    for k ← 0 to n – 1 do
        foreach edge (i, j) in E do
            if (d[i] + w(i, j) < d[j]) then
                dnew[j] ← d[i] + w(i, j);
            endif
        endfor
        // copia l'array dnew in d
        for int i ← 0 to n - 1 do
            d[i] ← dnew[i];
        endfor
    endfor
```

Il file `omp-sssp.c` contiene lo scheletro di un programma che implementa l'algoritmo di Bellman-Ford descritto sopra. Il programma accetta due parametri sulla riga di comando:

- Il primo parametro è il nome del file da cui leggere il grafo. Vengono forniti alcuni esempi: `rome99.gr` (porzione di mappa stradale di Roma, 3353 nodi e 8870 archi), `DE.gr` (porzione di mappa stradale del Delaware, 49109 nodi e 121024 archi), `VT.gr` (porzione di mappa stradale del Vermont, 97975 nodi e 215116 archi) e `ME.gr` (porzione di mappa stradale del Maine, 194505 nodi e 429842 archi). Suggerisco di iniziare con `rome99.gr` perché è quello più piccolo.
- Il secondo parametro è l'indice del nodo sorgente, che deve essere compreso tra 0 e n - 1 (default 0).

Al termine dell'esecuzione viene stampata la distanza tra il nodo sorgente e il nodo n - 1.
Completare la funzione `bellmanford()` seguendo lo pseudocodice descritto sopra. Iniziare con una versione seriale; se l'algoritmo è implementato correttamente, il programma dovrebbe stampare `d[3352]=30305.000000` per Roma, `d[49108]=69204.000000` per il Delaware e `d[97974]=129866.000000` per il Vermont; in tutti i casi si è usato il nodo 0 come sorgente.
Una volta che si ha una versione seriale funzionante, applicare i costrutti OpenMP appropriati per ottenerne una versione parallela; prestare attenzione a possibili race conditions.