Corso di High Performance Computing

Progetto 2016/2017

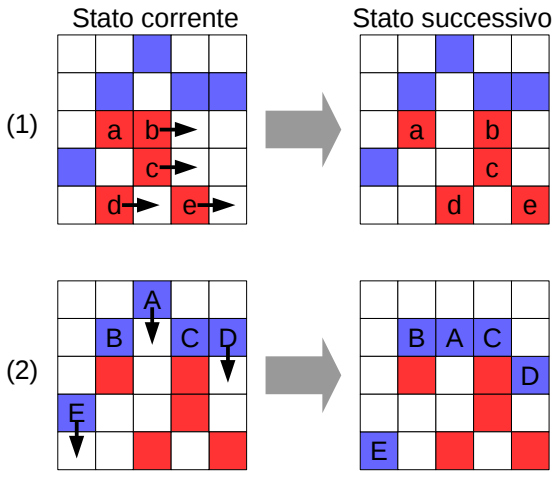
Moreno Marzolla

Versione 1.1 del 26/05/2017
Segnalata la necessità di gestire con attenzione le *ghost cell* (se usate)
Versione 1.0 del 24/05/2017
Prima versione di questo documento

Il modello Biham-Middleton-Levine

Il modello Biham-Middleton-Levine (BML) [1, 2] è un semplice automa cellulare bidimensionale usato per studiare flussi di traffico di veicoli in condizioni semplificate. Il modello consiste in una matrice quadrata di $N \times N$ celle. Ogni cella può essere in tre possibili stati: (*i*) vuota, (*ii*) contenente un veicolo che si muove da sinistra a destra (LR), (*iii*) contenere un veicolo che si muove dall'alto verso basso (TB). Il modello evolve a istanti discreti di tempo $t = 0, 1, 2, 3, \ldots$ Ogni passo di evoluzione consiste di due fasi da eseguire in sequenza. Nella prima fase, tutti i veicoli LR la cui cella di destra è vuota si spostano a destra di una posizione; nella seconda fase, i veicoli TB la cui cella in basso è vuota si spostano in basso di una posizione. Assumiamo un dominio ciclico, quindi i veicoli che escono da un lato rientrano dal lato opposto. Come sempre nel caso di automi cellulari si assume che tutti gli spostamenti avvengano contemporaneamente: per simulare ciò si fa uso di due domini, che rappresentano la situazione corrente e la situazione al passo successivo. I dati vengono sempre letti dalla griglia corrente e scritti nella griglia successiva.

La figura seguente mostra un esempio con una griglia 5×5; i veicoli che si muovono da sinistra a destra sono indicati in rosso, mentre quelli che si muovono dall'alto verso il basso sono indicati in blu.



In (1) si esegue la prima fase, che consiste nello spostamento verso destra di tutti i veicoli LR che hanno la cella a destra libera. I veicoli *b*, *c*, *d*, *e* possono spostarsi a destra di una posizione, mentre il veicolo *a* è bloccato da *b* nello stato corrente, e quindi non si sposta. Si noti che la decisione di ciascun veicolo se avanzare o no deve essere basata sull'analisi della griglia corrente. In (2) si esegue la seconda fase, che consiste nello spostamento verso il basso di tutti i veicoli TB che hanno la cella in basso libera. I veicoli *A*, *D*, *E* si spostano di una casella verso il basso, mentre *B* e *C* risultano bloccati da altri veicoli e non si spostano.

L'algoritmo da eseguire per realizzare la prima fase (spostamenti orizzontali) è descritto dallo

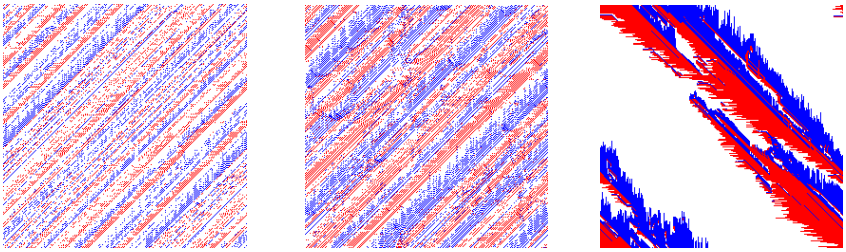
pseudocodice seguente. `cur` e `next` rappresentano la griglia corrente e quella successiva; gli elementi della griglia possono essere `EMPTY` (cella vuota), `LR` (veicolo con movimento sinistra-destra) oppure `TB` (veicolo con movimento alto-basso):

```
horizontal_step( cur, next )
    for i ← 0 to N - 1 do
        for j ← 0 to N - 1 do
            if ( cur[i][j-1] = LR and cur[i][j] = EMPTY ) then
                next[i][j] ← LR
            elseif ( cur[i][j] = LR and cur[i][j+1] = EMPTY ) then
                next[i][j] ← EMPTY
            else
                next[i][j] ← cur[i][j]
            endif
        endfor
    endfor
```

Ricordiamo che stiamo operando su domini ciclici, quindi ad esempio `cur[i][j-1]` dovrà essere `cur[i][N-1]` se $j = 0$; lo pseudocodice deve essere opportunamente adattato per gestire correttamente questa condizione. L'algoritmo per effettuare gli spostamenti in verticale è simile (con le dovute modifiche) a quello sopra; naturalmente, gli spostamenti in verticale dovranno essere effettuati sulla *nuova* griglia risultante *dopo* aver effettuato quelli orizzontali. Una volta effettuati gli spostamenti in verticale e in orizzontale si completa un passo dell'evoluzione dello stato dell'automa.

L'automa viene inizializzato definendo la dimensione N del dominio, la densità iniziale ρ di veicoli, e il numero di passi di simulazione da eseguire. Il parametro ρ è un valore reale compreso tra 0 e 1, e indica la probabilità che una cella risulti occupata ($\rho = 0$ produce un dominio vuoto, $\rho = 1$ posiziona un veicolo in ogni cella). Per ogni cella occupata si sceglie, con uguale probabilità, se piazzare un veicolo LR oppure TB; di conseguenza, in un dominio con $N \times N$ celle ci saranno in media $(\rho N^2)/2$ veicoli LR e $(\rho N^2)/2$ veicoli TB.

L'automa BML ha proprietà interessanti. Se la densità iniziale di veicoli è al di sotto di una soglia critica, i veicoli si auto-organizzano per muoversi senza interferire con altri veicoli (*free-flowing phase*). Se la densità, si possono formare degli ingorghi periodici che però si dissolvono consentendo il movimento dei veicoli. Se la densità supera una soglia critica, si forma un unico ingorgo permanente (*jammed phase*) e nessun ulteriore movimento è possibile. La figura seguente mostra lo stato dell'automa dopo 2048 iterazioni su una griglia 256×256 per diverse densità ρ di veicoli: $\rho = 0.2$, $\rho = 0.3$ e $\rho = 0.4$. Si noti come nel caso $\rho = 0.4$ si sia formato un ingorgo permanente.



$\rho = 0.2$ $\rho = 0.3$ $\rho = 0.4$

Sulla pagina del corso (http://moreno.marzolla.name/HPC/) è presente l'archivio `ProgettoHPC1617.tar.gz` contenente lo scheletro di una implementazione del modello BML. Il codice, che manca di alcune parti che devono essere implementate, può essere utilizzato come base per le versioni parallele da sviluppare (vedi seguito). È comunque consentita, qualsiasi modifica, inclusa la riscrittura da zero.

Cosa viene richiesto

Viene richiesto di consegnare due versioni parallele del programma:

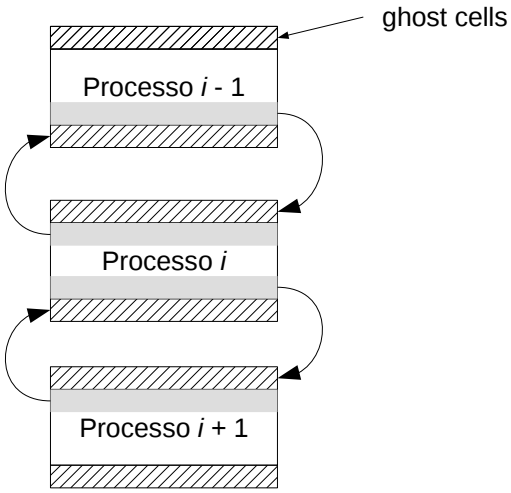
1. La prima, chiamata `omp-traffic.c`, deve sfruttare il parallelismo fornito da OpenMP; questa versione può essere basata sullo scheletro fornito, una volta completate le funzioni.
2. La seconda versione deve essere basata, a scelta, su MPI oppure CUDA (uno dei due, non entrambi). Il file deve chiamarsi rispettivamente `mpi-traffic.c` oppure `cuda-traffic.cu`.

Oltre ai due programmi di cui sopra, è obbligatorio includere:

3. Una relazione in formato PDF che descriva sinteticamente le strategie di parallelismo adottate e discuta la scalabilità ed efficienza dei programmi realizzati.

Suggerimenti

- Lo scheletro del programma fornito produce una singola immagine corrispondente allo stato dell'automa dopo K iterazioni. Sebbene sia possibile modificare il codice per salvare lo stato dell'automa dopo ogni iterazione, ad esempio per produrre un video, suggerisco di NON farlo mentre si misurano i tempi per il calcolo di speedup ed efficienza, in quanto il salvataggio dei file introduce dei ritardi che rendono le misure non significative.
- L'inizializzazione della griglia (la funzione `init()`) deve essere sempre effettuata da un singolo processo (il processo 0 nel caso di MPI, il master nel caso di OpenMP). Il motivo è che la funzione `rand()` della libreria C non è thread-safe, e pertanto non deve essere usata in contesti paralleli; inoltre il risultato dell'inizializzazione deve essere deterministico, e questo non sarebbe garantito nel caso di esecuzione concorrente.
- La versione OpenMP e MPI/CUDA devono produrre lo stesso risultato se vengono eseguite con gli stessi parametri di input, e se l'automa è inizializzato in modo deterministico da un unico processo. Suggerisco di confrontare i file di output prodotti dalle due versioni usando il comando Linux `cmp` per verificarne l'uguaglianza.
- Come sempre nel caso di domini ciclici può essere conveniente aumentare la griglia con righe e colonne aggiuntive (*ghost cell*) che vanno aggiornate ad ogni passo. In tal caso il dominio avrà dimensione effettiva pari a $(N + 2) \times (N + 2)$ celle. Il riempimento delle *ghost cell* va gestito con cura, perché l'automa BML presenta delle peculiarità che non erano presenti negli esercizi visti in laboratorio. In particolare, occorre tenere presente che la seconda fase di ogni passo di aggiornamento (spostamenti verticali) deve essere svolta sulla griglia risultante dopo la prima fase (spostamenti orizzontali).
- [MPI] Suggerisco di effettuare un partizionamento a blocchi per riga: se vengono lanciati P processi MPI, il processo 0 gestisce le prime N/P righe, il processo 1 le successive N/P righe e così via. Il master mantiene una copia dell'intero dominio, mentre ciascuno dei processi gestisce una partizione del dominio; le partizioni devono includere ghost cells che vanno aggiornate dopo ogni passo . La figura seguente mostra lo scambio delle righe sui bordi di un processo verso i processi vicini i - 1 e i + 1; nella figura non vengono mostrate le ghost cell laterali di ogni sottodominio, che potrebbero essere necessarie o meno a seconda dell'implementazione.



- [CUDA] Utilizzare *thread block* bidimensionali, in cui ogni thread calcola lo stato successivo di una cella del dominio. Ricordarsi che il numero massimo di thread per blocco supportato dal server CUDA usato per le esercitazioni è 512.

Ovviamente esistono diversi modi per realizzare le versioni MPI e CUDA. Suggerisco di iniziare con versioni semplici ma funzionanti; solo a questo punto, se si aspira ad una valutazione elevata, si può pensare di estendere e/o migliorare il codice.

Modalità di svolgimento del progetto

- Il progetto deve essere svolto individualmente. Non è consentito condividere, in tutto o in parte, il codice o la relazione relazione con altri studenti.
- È ammesso l'uso di porzioni di codice reperito in rete o tramite altre fonti purché (*i*) la provenienza di ogni frammento di codice scritto da terzi sia chiaramente indicata in un commento, e (*ii*) la licenza di tale codice ne consenta il riutilizzo. Il codice usato dal docente durante le lezioni o i laboratori può essere usato liberamente senza necessità di indicarne la fonte.
- I sorgenti devono essere adeguatamente commentati. All'inizio di ogni file deve essere indicato cognome, nome e numero di matricola dell'autore/autrice.
- Includere un file README contenente le istruzioni per la compilazione e l'esecuzione dei programmi consegnati; chi lo desidera può includere un Makefile per la compilazione (non obbligatorio).
- I programmi consegnati verranno compilati sui server utilizzati durante le esercitazioni; quindi, le versioni OpenMP e MPI verranno compilate su `disi-hpc.csr.unibo.it`, e le versioni CUDA verranno compilate su `disi-hpc-cuda.csr.unibo.it`. I programmi che non compilano o non eseguono correttamente su tali macchine non verranno presi in considerazione.
- Oltre ai sorgenti, è richiesto di consegnare una relazione in formato PDF denominata `Relazione.pdf`. La relazione non deve superare la lunghezza di sei facciate in formato A4, contando tutte le pagine (inclusi eventuali frontespizi). Il formato della relazione è libero, ma nell'archivio contente lo schema di progetto è presente uno schema in formato LibreOffice che chi lo desidera può usare. La relazione deve descrivere le implementazioni soffermandosi sulle strategie di parallelizzazione adottate, e studiare la scalabilità dell'algoritmo realizzato, mostrando i grafici di speedup ed efficienza. Nella relazione va

indicato il proprio cognome, nome e numero di matricola.

Modalità di consegna

Il progetto può essere consegnato in qualsiasi momento; è possibile consegnarlo prima o dopo aver sostenuto la prova scritta. L'eventuale valutazione positiva resta valida fino alla sessione d'esami di gennaio/febbraio 2018 inclusa; dopo tale data tutti i voti in sospeso (sia del progetto che delle prove scritte) verranno persi.

La consegna deve avvenire inviando una mail dal proprio indirizzo istituzionale @studio.unibo.it a moreno.marzolla@unibo.it con oggetto:

[HPC] Consegna Progetto 2016/2017

Nella mail vanno indicati **cognome**, **nome**, **numero di matricola** del mittente. Alla mail deve essere allegato un archivio in formato `.zip` oppure `.tar.gz` contenente i sorgenti e la relazione. L'archivio sarà denominato con il cognome e nome dell'autore (es., `MarioRossi.zip` oppure `MarioRossi.tar.gz`), e dovrà contenere una directory con lo stesso nome (es., `MarioRossi/`) contenente a sua volta i file secondo il seguente layout:

`MarzollaMoreno/src/omp-traffic.c`

`MarzollaMoreno/src/mpi-traffic.c` (*se si svolge la versione MPI*)

`MarzollaMoreno/src/cuda-traffic.cu` (*se si svolge la versione CUDA*)

`MarzollaMoreno/src/Makefile` (*facoltativo*)

`MarzollaMoreno/README`

`MarzollaMoreno/Relazione.pdf`

(*altri file che si ritengono utili*)

Valutazione dei progetti

Un progetto verrà considerato **sufficiente** se soddisfa almeno i seguenti requisiti minimi:

- I programmi compilano ed eseguono correttamente su istanze di input anche soggette a vincoli (es., dimensione dell'input multipla di...) sulle macchine usate per le esercitazioni (`disi-hpc.csr.unibo.it` per le versioni OpenMP e MPI, `disi-hpc-cuda.csr.unibo.it` per le versioni CUDA).
- La relazione dimostra un livello sufficiente di padronanza degli argomenti trattati.

Ulteriori aspetti che potranno comportare una valutazione superiore:

- Qualità del codice, in termini di efficienza, uso appropriato delle primitive e/o dei pattern di programmazione concorrente adeguati, generalità, eccetera.
- Qualità della relazione, in termini di chiarezza, correttezza e presentazione del contenuto; in particolare, verrà valutata positivamente la presenza nella relazione dei grafici di *speedup* ed efficienza con discussione dei risultati ottenuti. Non è richiesto di misurare i tempi di esecuzione sulle macchine usate per le esercitazioni. Verrà valutata la capacità di interpretare i risultati ottenuti piuttosto che i risultati di per se.

Checklist per la consegna

Prima di consegnare il progetto, assicurarsi che i requisiti fondamentali elencati nella lista seguente siano soddisfatti:

□	Vengono consegnate due versioni del programma, una che usa OpenMP, e una che usa (a scelta) MPI oppure CUDA.
□	I sorgenti compilano ed eseguono correttamente sui server utilizzati per le esercitazioni (`disi-hpc.csr.unibo.it` per le versioni OpenMP e MPI, `disi-hpc-cuda.csr.unibo.it` per le versioni CUDA).
□	La relazione è in formato PDF e ha lunghezza minore o uguale a 6 facciate.
□	I sorgenti e la relazione indicano chiaramente cognome, nome e numero di matricola dell'autore/dell'autrice.
□	Il progetto viene consegnato in un unico archivio .zip oppure .tar.gz, nominato con nome e cognome dell'autore, che include i sorgenti e la relazione in formato PDF.
□	La mail di consegna ha oggetto "[HPC] Consegna progetto 2016/2017"; la mail viene spedita dal proprio indirizzo di posta istituzionale, e include cognome, nome e numero di matricola dell'autore/autrice del progetto.

Riferimenti bibliografici

[1] O. Biham, A. Alan Middleton, D. Levine, *Self-organization and a dynamical transition in traffic-flow models*. Phys. Rev. A 46 (10), nov 1992: doi:10.1103/PhysRevA.46.R6124. ISSN 1050-2947. Disponibile all'indirizzo https://arxiv.org/abs/cond-mat/9206001

[2] *Biham-Middleton-Levine traffic model*, in Wikipedia, Retrieved May 20, 2017, https://en.wikipedia.org/wiki/Biham%E2%80%93Middleton%E2%80%93Levine_traffic_model