

# Confronto delle prestazioni CPU-GPU per la simulazione di un reticolo di Ising-2D

Stefano Mandelli  
Matricola: 861838

## Introduzione

Per confrontare le prestazioni di CPU e GPU in questo lavoro è stato scelto il modello di Ising 2D. È un modello analitico, nel senso che al limite termodinamico esiste una soluzione analitica esatta. È stato possibile riportare il codice, operazione per operazione (compresa la generazione dei numeri pseudorandom) da singola CPU alla fine parallelizzazione delle GPU, in questo modo sarà possibile valutare lo speed-up di prestazioni in modo quantitativo perfettamente consistente. Se si fosse usato un generatore di numeri pseudorandom diverso, la comparazione sarebbe stata meno consistente in quanto si sarebbero andati a comparare risultati ottenuti con implementazioni diverse. I punti che verranno trattati in questo lavoro sono i seguenti:

- Breve introduzione sul modello usato;
- Implementazione del codice;
- Consistenza fisica del modello;
- Discussione sulla diversità dei risultati;
- Comparazione delle prestazioni per diverse implementazioni;

## 1 Scelta del modello

Il modello di Ising-2D è un modello che si presta molto bene ad essere parallelizzato, in quanto è un tipico modello interagente a corto range. In questo modo

è possibile pensare ad una strategia di parallelizzazione efficace. Brevemente, il modello di Ising è caratterizzato da un reticolo (ad esempio ipercubico) in  $D$  dimensioni. Ad ogni cella del reticolo viene associato uno spin  $s_i$  che può essere solo del tipo  $s_i = \{+1, -1\}$  a seconda che la direzione del dipolo magnetico (o spira), associato alla cella  $i$  –esima del reticolo risulti verso l'alto o verso il basso. Il sistema è descritto dall'Hamiltoniana di Ising

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} s_i s_j - h \sum_i s_i \quad (1)$$

dove  $h$  identifica un eventuale campo magnetico uniforme esterno. La prima sommatoria è fatta su tutte le coppie di siti primi vicini e  $J$  indica la costante di accoppiamento tra spin. In questo lavoro si considera il caso  $D = 2$  dimensionale,  $J > 0$  e campo magnetico esterno nullo, quindi  $h = 0$ . Al limite termodinamico, il modello di Ising (escluso il caso  $1D$ ) presenta una transizione di fase in prossimità di una temperatura critica  $T_c$ . Per temperature maggiori di  $T_c$  il sistema si comporta in modo paramagnetico. Per temperature inferiori invece si ha un fenomeno di magnetizzazione spontanea. Le principali grandezze fisiche che possono essere calcolate sono la magnetizzazione media del sistema  $\langle M \rangle$  e la capacità termica a volume costante  $\langle C_V \rangle$  che viene calcolata col teorema di fluttuazione-dissipazione. Tutte queste quantità estensive sono divergenti nel limite termodinamico. Nelle simulazioni statistiche vengono considerati dei modelli finiti di  $N = D \times D$  spins che si desidera confrontare per diversi size. È utile pertanto confrontare la densità di calore specifico  $c_V = CV/N$  e di magnetizzazione che è la variabile coniugata al campo esterno  $h$

$$m = \frac{1}{N} \langle \sum_i s_i \rangle \quad (2)$$

Per il caso  $h = 0$  e  $T > T_c$ ,  $\langle M \rangle$  si annulla. Questo comportamento può essere spiegato nel seguente modo: per  $T > T_c$  le fluttuazioni termiche prevalgono sulla tendenza del termine di interazione  $J$  ad allineare gli spin in un'unica direzione. La lunghezza di correlazione è molto piccola e ogni spin ha la stessa probabilità di avere come valore  $+1$  o  $-1$ , in questo modo  $\langle M \rangle$  risulta nulla. Per temperature  $T < T_c$  gli spin risentono fortemente dell'interazione coi loro primi vicini. Si nota che in questo range di temperature il modello di Ising 2D, presenta una transizione di fase netta. Il sistema passa, in modo spontaneo, da una situazione di disordine in cui gli spin sono orientati prevalentemente nella stessa direzione quindi hanno in maggioranza valore  $+1$  oppure  $-1$ . I due casi, per campo magnetico esterno nullo ( $h = 0$ ), sono equiprobabili, in quanto per  $h = 0$  l'Hamiltoniana di Ising è pari per inversione di tutti gli spins, per tempi molto lunghi entrambi gli stati vengono popolati per la stessa quantità di tempo facendo risultare, anche

in questo caso,  $\langle M \rangle = 0$ . Per modelli finiti sufficientemente grandi è ugualmente possibile effettuare delle valutazioni e misure di magnetizzazione media spontanea del sistema per  $T < T_c$ . Questo si ottiene settando lo stato di partenza in modo tale che uno dei due stati sia più popolato dell'altro. Se per convenzione si sceglie uno stato di partenza con una maggioranza di spins a valore  $+1$ , lo stato di equilibrio dell'Hamiltoniana di Ising, con molti spin a valore  $+1$  sarà favorito. In questo modo, per temperature inferiori a quella critica, si hanno dei valori di magnetizzazione non nulli e dello stesso segno ed è quindi possibile valutare l'entità della magnetizzazione spontanea.

## 1.1 Algoritmo di Metropolis

Dato che le fluttuazioni di energia, rispetto all'energia totale del sistema, sono piccole, un modo sufficientemente efficiente di generare delle mosse Monte Carlo, è quello definito dalla *dinamica a singolo spin-flip*. I vari stati vengono generati in modo che il successivo sia differente dal precedente per il flip di un singolo spin del reticolo preso inizialmente in modo casuale. Lo stato  $\mu$  e quello  $\nu$  differiscono tra loro solo per il flip di un singolo spin. In questo modo è possibile definire la probabilità di selezione come

$$g(\mu \rightarrow \nu) = \frac{1}{N}. \quad (3)$$

Con questa probabilità di selezione, la condizione del detailed balance prende la forma

$$\frac{P(\mu \rightarrow \nu)}{P(\nu \rightarrow \mu)} = \frac{g(\mu \rightarrow \nu)A(\mu \rightarrow \nu)}{g(\nu \rightarrow \mu)A(\nu \rightarrow \mu)} = \frac{p_\nu}{p_\mu} = e^{-\beta(E_\nu - E_\mu)}. \quad (4)$$

Dato che  $g(\mu \rightarrow \nu) = g(\nu \rightarrow \mu)$  si sceglie l'acceptance ratio in modo tale che soddisfi l'equazione (4)

$$\frac{A(\mu \rightarrow \nu)}{A(\nu \rightarrow \mu)} = e^{-\beta(E_\nu - E_\mu)}, \quad (5)$$

da cui è possibile dedurre che

$$A(\mu \rightarrow \nu) = A_0 e^{-\frac{1}{2}\beta(E_\nu - E_\mu)}. \quad (6)$$

Per avere un algoritmo che sia il più efficiente possibile l'acceptance ratio deve essere significativamente diversa da zero. Il parametro  $A_0$  è scelto in funzione ad alcune considerazioni su come è fatta l'Hamiltoniana di Ising. E' facile osservare che la differenza di energia tra due stati, in modulo, è al massimo pari a  $|\Delta E| = 2zJ$  dove  $z$  è il numero di primi vicini, che nel caso di reticolo 2D vale  $z = 4$ ,

quindi per il reticolo 2D abbiamo che al massimo  $|\Delta E| = 8J$ . La differenza di energia tra lo stato  $\mu$  e  $\nu$  è

$$|E_\nu - E_\mu| \leq 2zJ. \quad (7)$$

In questo modo, il massimo possibile valore dell'esponentiale vale

$$e^{-\frac{1}{2}\beta(E_\nu - E_\mu)} \leq e^{\beta zJ} \quad (8)$$

che permette di stabilire la scelta migliore possibile del coefficiente

$$A_0 = e^{-\beta zJ}. \quad (9)$$

La scrittura finale per l'acceptance ratio risulta quindi:

$$A(\mu \rightarrow \nu) = e^{-\frac{1}{2}\beta(E_\nu - E_\mu + 2zJ)}, \quad (10)$$

in modo da avere  $A(\mu \rightarrow \nu) \leq 1$ . Si può verificare che l'  $A(\mu \rightarrow \nu)$  scritta ora è molto inefficiente. Il sistema rimane per troppo tempo nello stesso stato. Una scelta migliore che rispetta tutte le condizioni di quella precedente è data proprio dall'acceptance ratio proposta da Metropolis

$$A(\mu \rightarrow \nu) = \begin{cases} e^{-\beta(E_\nu - E_\mu)} & E_\nu - E_\mu > 0 \\ 1 & \text{altrimenti.} \end{cases} \quad (11)$$

Per il modello di Ising le acceptance ratio appena descritte hanno la particolarità che possono essere calcolate mediante la sola conoscenza degli spins primi vicini allo spin di cui si propone il suo flip, questo perchè l'interazione nel modello di Ising è a corto raggio. Nel caso dell'Hamiltoniana di Ising, questa differenza è possibile scriverla in modo molto semplice, in funzione solo dagli spin primi vicini dello spin di cui si propone il flip:

$$\begin{aligned} E_\nu - E_\mu &= -J \sum_{\langle i,j \rangle} s_i^\nu s_j^\nu + J \sum_{\langle i,j \rangle} s_i^\mu s_j^\mu = \\ &= -J \sum_{i \neq k} s_i^\nu (s_k^\nu - s_k^\mu) = -2J s_k^\mu \sum_{i \neq k} s_i^\mu. \end{aligned} \quad (12)$$

## 2 Implementazione del codice

La strategia di implementazione che è stata adottata in questo lavoro consiste in una prima implementazione su CPU e la successiva implementazione su scheda grafica. L'obiettivo è quello di riottenere gli stessi risultati e gli stessi grafici.

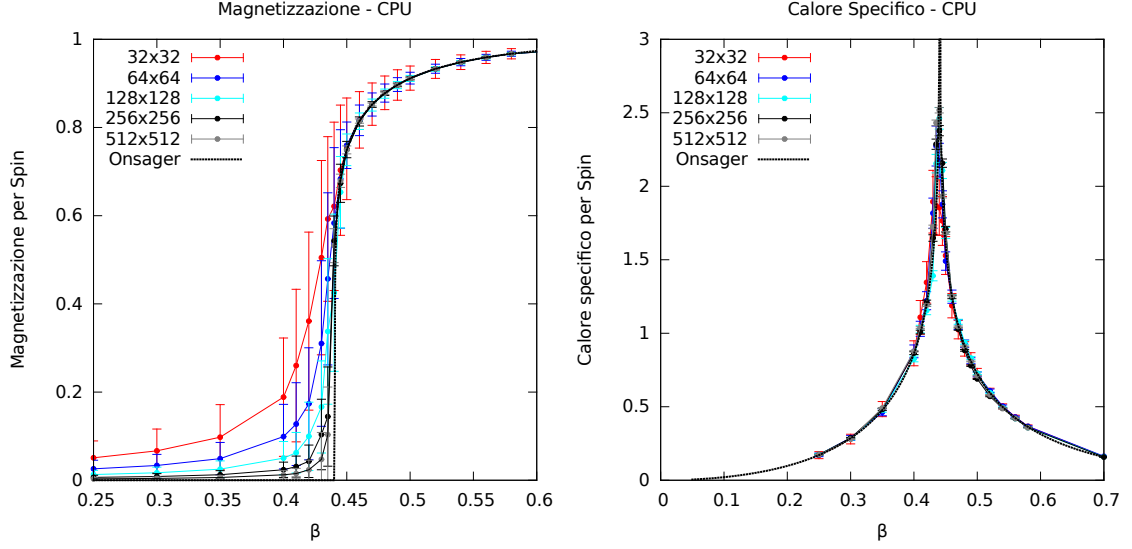


Figura 1: *Magnetizzazione e Calore Specifico* per diverse taglie di reticolo, in funzione di  $\beta$  generati su CPU

Un primo confronto viene fatto con i grafici di *Magnetizzazione e Calore Specifico* in funzione di  $\beta$ , con la soluzione esatta di Onsager. Questo confronto è per verificare la compatibilità della simulazione con il modello fisico. Successivamente sono stati confrontati gli stessi risultati ottenuti su GPU e CPU.

Il secondo risultato che viene presentato riguarda le caratteristiche (punti favorevoli e punti sfavorevoli) di diversi PRNG, nel particolare è stato confrontato il numero di step di termalizzazione in funzione alla complessità numerica della generazione.

Il terzo risultato che viene presentato riguarda la ricerca di un BLOCKL di dimensione tale da portare a saturazione ogni singolo WARP.

Le conclusioni finali sull'ottimizzazione del programma terranno conto di tutte le conclusioni presentate precedentemente.

## 2.1 Implementazione su CPU

In Fig. 1 sono presentati i risultati di *Magnetizzazione e Calore specifico*, per diversi size in funzione di  $\beta$ , comparati con la soluzione analitica al limite termodinamico di Onsager.

Si notano in modo evidenti gli effetti di *size scaling* per reticoli molto piccoli. Al crescere della taglia del reticolo, i risultati della simulazione si avvicinano sempre di più al modello al limite termodinamico.

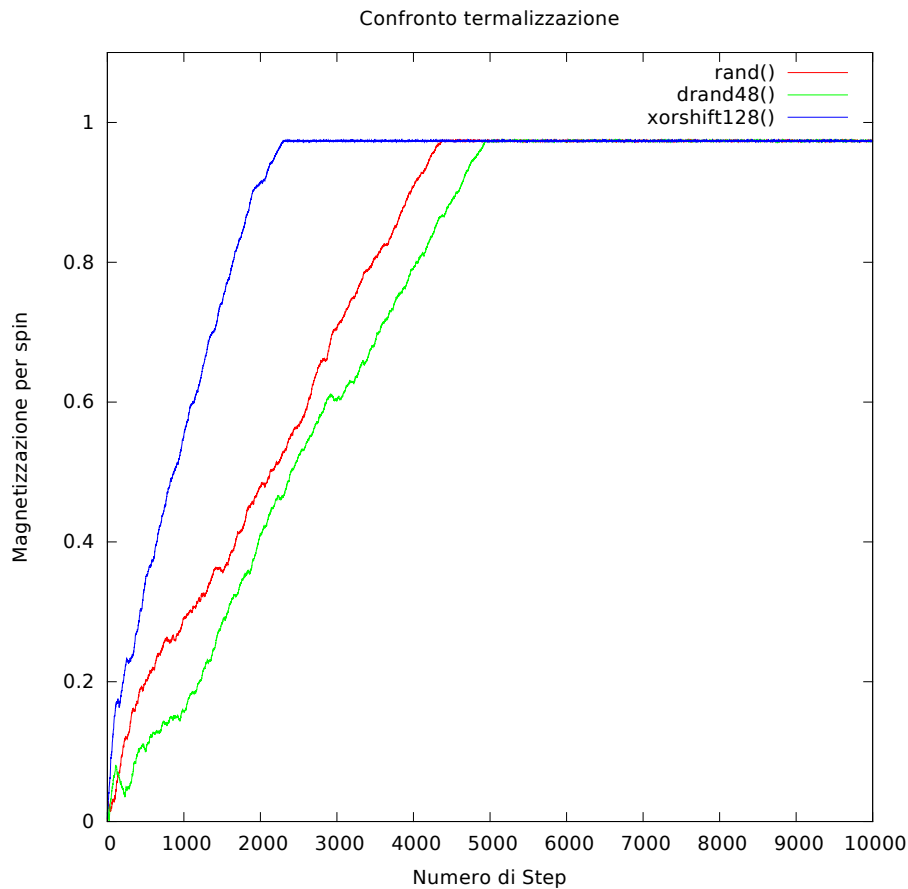


Figura 2: *Passi Montecarlo* necessari per raggiungere la termalizzazione in funzione a tre diversi PNRG.

## Comparazione tra diversi PNRG su CPU

In Fig. 2 viene presentato il confronto tra tre diversi PNRG. Come si nota, diversi generatori fanno termalizzare il sistema in un numero molto diverso di step. Tra i passi di termalizzazione del generatore più performante e quello meno performante c'è circa un fattore quattro.

## 2.2 Implementazione su GPU

L'implementazione del codice passa attraverso due grandi fasi. La prima consiste nell'organizzare il reticolo di Ising sulla GPU. L'obiettivo che ci si pone è quello di usare nel modo migliore le aree di memorie in modo da garantire sempre coalescenza, velocità di lettura/scrittura e minimizzazione di bank conflict. La coalescenza consiste nell'effettuare chiamate ad aree di memoria allineate in modo da massimizzare le performance. La velocità di lettura e scrittura dipende dal tipo

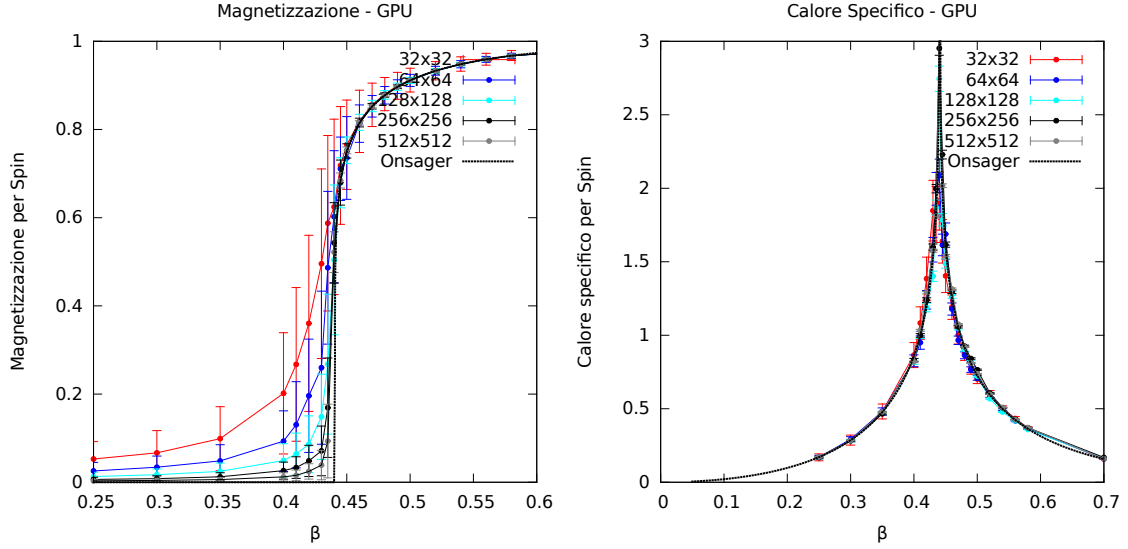


Figura 3: *Magnetizzazione e Calore Specifico* per diverse taglie di reticolo, in funzione di  $\beta$  generati su GPU

di memoria usata mente il problema del bank conflict consiste nel fatto che due aree di memoria non possono essere lette contemporaneamente. Se nel programma ci sono chiamate di questo tipo, vengono schedate in modo sequenziale perdendo tutto il vantaggio della parallelizzazione.

Una seconda fase consiste nel riportare, operazione per operazione, il generatore di numeri pseudorandom usato nel caso per CPU, in parallelo su GPU. Questo tipo di passaggio non è banale in quanto la generazione di numeri pseudorandom è per definizione seriale. La generazione del numero  $n + 1$  dipende dal numero  $n$ . Dato che l'evoluzione del nostro modello è di tipo *Metropolis* e i numeri random sono quindi una necessità principe, per non perdere parallelizzazione del codice è necessario elaborare una strategia di estrazione parallela e performante per le GPU.

## Implementazione del modello

Il reticolo di Ising è stato posizionato sulla griglia della GPU utilizzando la *memoria shared* per ottenere migliori performance. La memoria shared è divisa in blocchi indipendenti tra loro, questo fa sì che il reticolo iniziale, debba essere prima posizionato nella memoria shared e successivamente vanno inserite le condizioni di raccordo blocco/blocco e quindi effettuare l'update degli spin usando una doppia scacchiera. Il primo livello di scacchiera, a blocchi grandi (relativi ai blocchi della shared memory), il secondo livello di scacchiera, a blocchi piccoli nel

sensò che l'update dei siti viene svolto a scacchiera anche tra i threads all'interno dello stesso blocco. Tutti i threads dello stesso blocco condividono la stessa area di memoria shared. In Fig. 4 sono stati plottati il tempo per proporre un update in funzione al size del reticolo.

### 2.3 Implementazione del PRNG

Una classe di generatori efficienti è stata proposta da Marsaglia ed è quella dei generatori **xorshift**. Consistono nel scegliere in modo opportuno, un vettore di semi iniziali in cui le componenti sono bits. Se si sceglie un vettore  $x$  di lunghezza  $w$ , è possibile dimostrare che si ottiene un ottimo e molto veloce generatore di numeri pseudo-random effettuando su questo vettore operazioni di shift. Definiamo la matrice  $\mathbf{L}$

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 \end{pmatrix} \quad (13)$$

come matrice di *spostamento a sinistra* che trasforma il vettore  $x = (x_1, \dots, x_w)$  in  $x = (x_2, \dots, x_w, 0)$ . L'operazione totale di XORShift proposta da Marsaglia è però di or esclusivo quindi per effettuare uno xorshift di  $a$  posizioni si ha che

$$x_{n+1} = x_n(\text{Id} \oplus \mathbf{L}^a). \quad (14)$$

Con il simbolo  $\oplus$  si intende la somma con mod 2 su ogni singolo elemento. Un vettore del tipo  $x_n = (1, 1, 0, 1)$  viene trasformato con uno xorshift sinistro di  $a = 1$  nel vettore  $x_{n+1}$  diventando

$$x_{n+1} = x_n(\text{Id} \oplus \mathbf{L}^1) = (1, 1, 0, 1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} = (0, 1, 1, 1). \quad (15)$$

Questa operazione, per quanto riguarda il linguaggio di programmazione C, può essere semplicemente svolta tramite due operazioni di bitwise. Dal punto di vista del codice, l'equazione (14) risulta molto semplice  $\mathbf{x} = \mathbf{x} \wedge (\mathbf{x} \ll \mathbf{a})$ . In letteratura viene consigliato di usare quattro step di shift. L'algoritmo implementato è quindi

$$x_{n+1} = x_n(\text{Id} \oplus \mathbf{L}^a)(\text{Id} \oplus \mathbf{R}^b)(\text{Id} \oplus \mathbf{L}^c)(\text{Id} \oplus \mathbf{R}^d) \quad (16)$$

dove  $\mathbf{R}$  è la matrice di *spostamento a destra* ed è definita come  $\mathbf{R} = \mathbf{L}^T$ . I parametri  $a, b, c, d$  sono le posizioni di shift rispettivamente a sinistra, destra, sinistra



e ancora a destra. È stato dimostrato che per valori opportuni dei parametri  $a, b, c, d$ <sup>1</sup> si hanno sequenze di numeri pseudo-random con periodo di  $2^w - 1$ . In questo caso  $w = 128$  quindi la catena di numeri è molto lunga ed è adeguata per essere utilizzata in simulazioni lunghe. La generazione in parallelo di numeri pseudo-random, avviene tramite l’allocazione in memoria di quattro vettori contenenti i seed iniziali. La scelta del seed ha un ruolo fondamentale tutto questo. Ad ogni nodi di calcolo (nel nostro caso delle GPU quindi ad ogni threads) deve esserci associato il suo quartetto di seed iniziali in modo tale che le varie sottocatene associate ai vari threads non si sovrappongono. Se ciò si verificasse, si perderebbe moltissima statistica perchè le catene associate ad ogni threads sarebbero praticamente tutte molto simili, si introdurrebbe quindi un forte accoppiamento dovuto alla totale (o anche parziale) perdita della statisticità. Per evitare questo, la catena totale di numeri PR viene spezzata in punti ben precisi, in particolar modo vengono presi valori di  $a, b, c, d$  ogni 10000 generazioni (numero di step Monte Carlo della simulazione) e vengono assegnati ad ogni threads. In questo modo ogni thread genera catene distanti 10000 numeri, esattamente il numero di step Monte Carlo, escludendo ogni tipo di sovrapposizione tra le catene.

### 3 Consistenza del modello Fisico

Nelle Figure [1] e [3] sono raffigurati tre grafici. Il primo rappresenta la magnetizzazione (per spin) in funzione di  $\beta$ , il secondo il calore specifico per spin, sempre in funzione di  $\beta$  e nel terzo viene fatto un fit dei valori di magnetizzazione, per ricavare l’esponente critico  $\alpha$ . Nell’intorno del  $\beta_c$  la magnetizzazione si comporta come

$$M(\beta) \sim \left| 1 - \frac{\beta_c}{\beta} \right|^\alpha. \quad (17)$$

Effettuando un fit dei valori intorno alla temperatura critica, mettendo come parametri liberi  $\alpha$  e  $\beta_c$  è quindi possibile ricavare il valore dell’esponente critico e della temperatura critica di transizione di fase. Nelle immagini 1 e 3 sono riportati i valori del fit trovati che sono compatibili con quelli noti.

### 4 Discussione sulla diversità dei risultati

In Tabella 1 sono riportati i valori di magnetizzazione per un reticolo  $32 \times 32$ , con relativo errore, in funzione di  $\beta$ . Come detto in precedenza, si è riusciti

---

<sup>1</sup>Marsaglia nel suo articolo consiglia  $a = 20, b = 11, c = 27, d = 6$

a portare completamente, operazione per operazione, l'algoritmo usato su CPU alle schede grafiche, quindi ci si aspettava che dopo lo stesso numero di passi di termalizzazione e di step Monte Carlo, il dato sia proprio identicamente lo stesso. Come si può notare in tabella i dati, seppur compatibili tra loro all'interno del proprio errore, non sono identicamente lo stesso dato. La spiegazione consiste nel modo in cui vengono estratti i numeri pseudocasuali. Nel caso dell'CPU, in modo seriale, l'evoluzione di uno spin è contraddistinto da un certo numero PR della catena e così via in modo ordinato su tutta l'estensione del reticolo, per tutti gli step Monte Carlo usati. Nel caso della GPU, la catena di numeri pseudo random viene suddivisa in sottocatene. Il numero pseudorandom che caratterizzava l'evoluzione dello spin nominato precedentemente, sulla GPU può caratterizzare un altro spin. È quindi corretto ritrovarsi con valori differenti, ma molto vicini tra loro e compatibilissimi all'interno del loro errore statistico. In conclusione i dati ci stanno dicendo è che nonostante ci sia un rimescolamento dei numeri casuali, la statistica del sistema rimane sempre la stessa, quindi i due dati sono perfettamente compatibili.

## 5 Comparazione delle prestazioni

Il codice è stato implementato in CPP per singola CPU, ed in cuda C con due varianti, la prima con il reticolo tutto su global memory la seconda utilizzando l'apporto anche della memoria shared. I risultati sono riportati in Figura 4. Come è possibile notare, una volta saturato il sensore, lo speed-up raggiunto è nell'ordine dei 2 ordini di grandezza rispetto allo stesso codice girato su CPU. L'utilizzo della memoria shared non ha velocizzato il processo. Le migliori prestazioni sono state ottenute aggiornando il reticolo a scacchiera sulla memoria globale.

$\beta$	$\langle M \rangle$ -GPU	$\sigma_{\langle M \rangle-GPU}$	$\langle M \rangle$ -CPU	$\sigma_{\langle M \rangle-CPU}$
0,250000	0,052712	0,039474	0,050932	0,038267
0,300000	0,066671	0,050460	0,066645	0,049545
0,350000	0,098850	0,072529	0,097678	0,073882
0,400000	0,201638	0,137473	0,188406	0,134147
0,410000	0,267280	0,174247	0,260063	0,172984
0,420000	0,360316	0,199749	0,360821	0,201873
0,430000	0,495578	0,214933	0,504824	0,220384
0,435000	0,587416	0,199017	0,592545	0,186706
0,440000	0,624490	0,198886	0,621196	0,190946
0,445000	0,718453	0,133490	0,703127	0,147649
0,450000	0,765466	0,101283	0,751676	0,115272
0,460000	0,818402	0,070646	0,817304	0,063977
0,470000	0,855991	0,049240	0,853003	0,048010
0,480000	0,880319	0,037618	0,878726	0,038106
0,490000	0,899063	0,030522	0,896382	0,034484
0,500000	0,912002	0,026942	0,912000	0,027388
0,520000	0,933304	0,020621	0,932910	0,021438
0,540000	0,948533	0,016265	0,948100	0,016604
0,560000	0,959347	0,013461	0,958931	0,013762
0,580000	0,967683	0,011340	0,967213	0,011583
0,700000	0,990200	0,005175	0,990184	0,005217

Tabella 1: Valori di magnetizzazione con il loro errore.

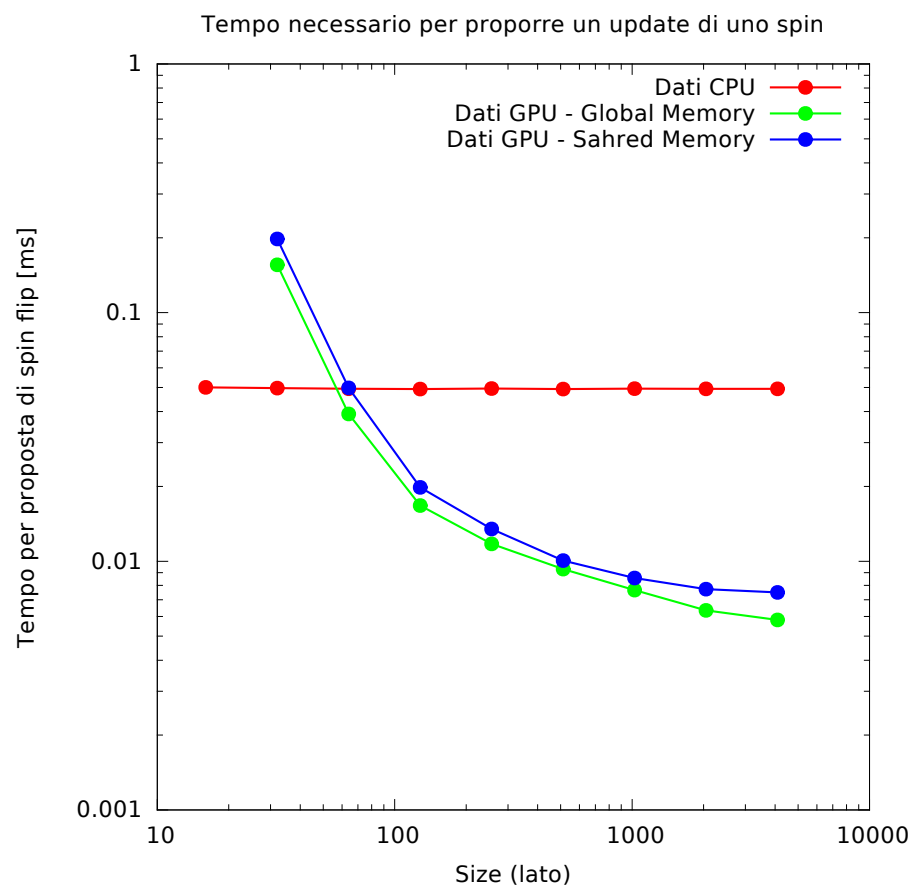


Figura 4: CPU/GPU