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

Stefano Mandelli
Matricola: 861838

13 - Luglio - 2016

Introduzione

In questo lavoro vengono messe a confronto le prestazioni di CPU[1] e GPU[2], usando come banco di prova un modello fisico ben conosciuto di cui sono note, in modo analitico, le espressioni delle varie grandezze fisiche che si vogliono simulare. Il modello in questione è il *reticolo di Ising bidimensionale*.

Durante l'implementazione da CPU a GPU, è stato possibile riportare il codice, da un ambienete all'altro, mantenendo lo stesso algoritmo e quindi lo stesso numero di operazioni che vengono fatte durante l'aggiornamento dei siti reticolari.

È stato possibile riportare da un ambiete all'altro anche il generatore di numeri pseudorandom, in modo da rendere consistente la valutazione dello speed-up. Se si fosse usato un generatore di numeri pseudorandom diverso, la comparazione sarebbe stata correlata anche al tipo di implementazione del *Generatore di Numeri Pseudorandom (PNRG)*.

Il lavoro consisterà in una prima presentazione del modello fisico usato a cui seguirà la presentazione dell'algoritmo usato per l'update dei siti reticolari. Una volta presentati modello e algoritmo di update viene descritta l'implementazione su GPU discutendo in primis la consistenza fisica della simulazione e successivamente i vari accorgimenti per ottimizzare le prestazioni.

[1] Central Processing unit
[2] Graphics Processing Unit

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. Per questo tipo di interazioni è facile trovare strategie di parallelizzazione efficaci.

Il modello di Ising è caratterizzato da un reticolo in D dimensioni. Ad ogni cella del reticolo viene associato uno spin che può essere solo del tipo $s_i = \{+1, -1\}$ a seconda che la direzione del dipolo magnetico, associato alla cella $i - esima$ del reticolo, risulti verso l'alto o verso il basso.

Il sistema è descritto dall'Hamiltoniana di Ising

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

dove h identifica un eventuale campo magnetico uniforme esterno. La prima sommatoria è su tutte le coppie di siti primi vicini e J indica la costante di accoppiamento tra spin.

In questo lavoro si considera il caso il caso $D = 2$ dimensionale, $J > 0$ e a 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 è perfettamente *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 divergono al limite termodinamico.

Nelle simulazioni statistiche vengono considerati dei modelli finiti di $N = L \times L$ 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

$$m = \frac{1}{N} \left\langle \sum_i s_i \right\rangle. \tag{2}$$

Per il caso $h = 0$ e $T > T_c$, $\langle M \rangle$ si annulla. Questo comportamento può essere spiegato a causa del fatto che 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 diventa 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. Scendendo con la temperatura sotto a T_c, il sistema passa in modo spontaneo, da una situazione totalmente disordinata, ad una ordinata in cui gli spin sono in gran parte allineati nella stessa direzione. I due casi (tutti spin *up* o tutti spin *down*), 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 possibile effettuare delle valutazioni e misure di magnetizzazione media spontanea del sistema per $T < T_c$ che abbiano valore non nullo. 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.

1.1 Algoritmo di Metropolis

L'algoritmo usato per far evolvere il sistema è quello di *Metropolis*, con aggiornamento del sito per *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 in modo casuale, quindi lo stato μ e quello ν differiscono tra loro solo per il flip di un singolo spin.

Si denisce quindi l'algoritmo di Metropolis per un reticolo di Ising, caratterizzato da aggiornamento a singolo spin flip, tramite la definizione di una probabilità di selezionare lo spin da aggiornate, chiamata *probabilità di selezione* $g(\mu \to \nu)$ e una probabilità di accettare l'update proposto, chiamata *probabilità di accettazione* $A(\mu \to \nu)$. Essendo la dinamica a *singolo spin flip* si ha che la probabilità che uno spin ha di essere selezionato è la stessa per tutti gli spin. Dato N il numero totale di spin nel reticolo, si ha che

$$g(\mu \to \nu) = \frac{1}{N}\,. \tag{3}$$

Con la probabilità di selezione definita in questo modo la condizione del *bilancio dettagliato* prende la forma

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

e dato che $g(\mu \to \nu) = g(\nu \to \mu)$ si sceglie il *rateo di accettazione* in modo tale che soddisfi l'equazione (4)

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

da cui è possibile dedurre che

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

Per avere un algoritmo che sia il più efficiente possibile il *rateo di accettazione* deve essere significativamente diverso da zero. Il parametro A_0 è scelto in funzione ad alcune considerazioni relative alla struttura dell'Hamiltoniana di Ising. È 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. 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 μ e ν è

$$|E_\nu - E_\mu| \leq 2zJ. \tag{7}$$

In questo modo, il massimo valore possibile dell'esponenziale è

$$e^{-\frac{1}{2}\beta(E_\nu - E_\mu)} \leq e^{\beta z J} \tag{8}$$

che permette di stabilire la scelta migliore possibile del coefficiente

$$A_0 = e^{-\beta z J}\,. \tag{9}$$

La scrittura finale per il *rateo di accettazione* risulta quindi essere

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

in modo da avere $A(\mu \to \nu) \leq 1$. Si può verificare che l' $A(\mu \to \nu)$ 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 quella in Fig. 1, data proprio dall'acceptance ratio proposta da Metropolis

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

Per il modello di Ising i *ratei di accettazione* appena descritti hanno la particolarità che possono essere calcolati 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.

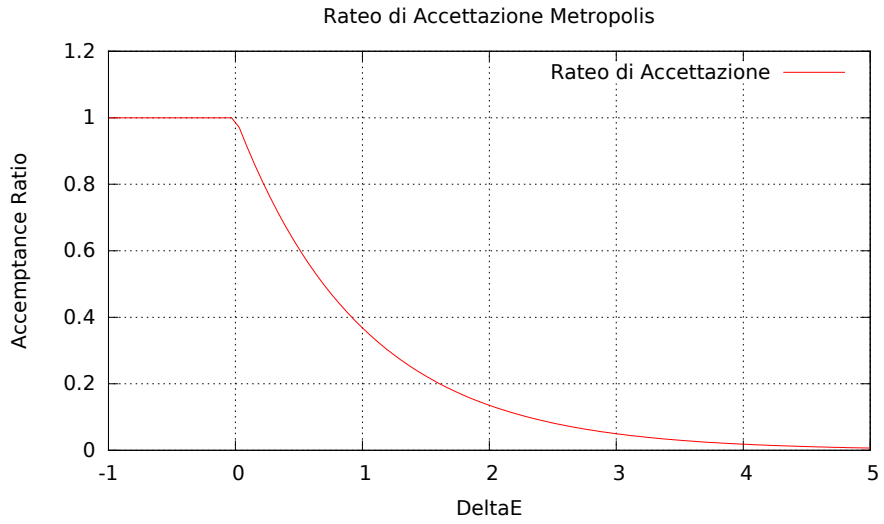

Figura 1: Rateo di Accettazione Metropolis in funzione a ΔE

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

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

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 fisici e valutare lo *speed-up* dato dall'utilizzo della GPU.

La consistenza fisica della simulazione viene fatta comparando i grafici di *Magnetizzazione* e *Calore Specifico* in funzione di β, con la soluzione esatta di Onsager al limite Termodinamico. 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. Tenendo in considerazione il fatto che in un blocco possono operare in modo parallelo 1024 threads è necessario trovare il numero corretto di threads in x e y in modo da avvicinarsi il più possibile a 1024.

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

3 Implementazione su CPU

In Fig. 2 sono presentati i risultati di *Magnetizzazione* e *Calore specifico*, per diversi size in funzione di β, comparati con la soluzione analitica al limite termodinamico di Onsager.

Si notano in modo evidente 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.

I risultati riportati in Fig. 2 mostrano anche che l'implementazione dell'algoritmo di update scritto precedente restituisce, su CPU, i valori attesi. Verificata la compatibilità tra i valori numerici simulati e quelli teorici della soluzione di Onsager, è ora possibile passare all'implementazione su GPU con l'obiettivo di ottenere gli stessi risultati numerici valutando il relativo *speed-up*.

In Fig. 3 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 genratore più performante e quello meno performante c'è circa un fattore tre.

4 Implementazione su GPU

L'implementazione del codice su GPU è stata organizzata in modo da poter mettere in luce tutte le varie caratteristiche che contraddistinguono le *schede grafiche*. La prima caratteristica è l'elevato numero di ALU[3] messo a disposizione rispetto alle CPU, quindi una prima implementazione del codice è stata fatta portando il reticolo sulla *memoria globale* della scheda e cercando di saturare il numero di ALU impegnate in modo parallelo, con una strategia di update a scacchiera.

La seconda caratteristica di questo tipo di devices è la possibilità di ottimizzare il proprio programma usando tutti i vari *livelli di memoria* che vengono

[3] Arithmetic and Logic Unit

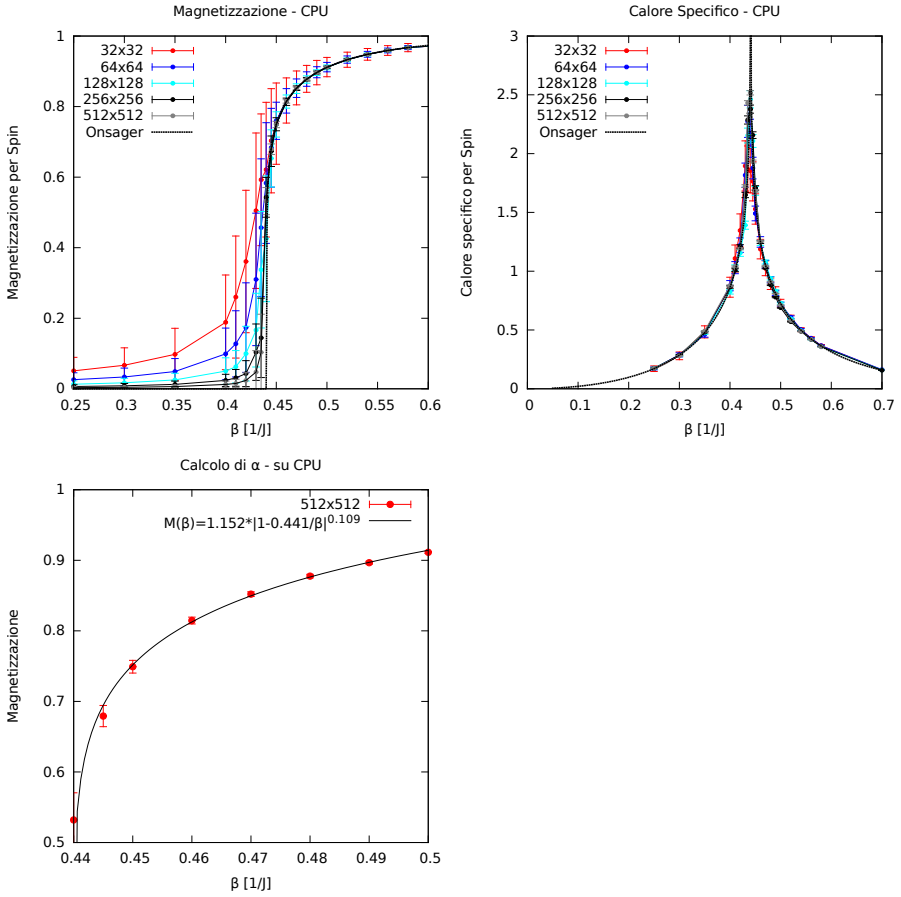

Figura 2: *Magnetizzazione* e *Calore Specifico* per diverse taglie di reticolo, in funzione di β generati su CPU

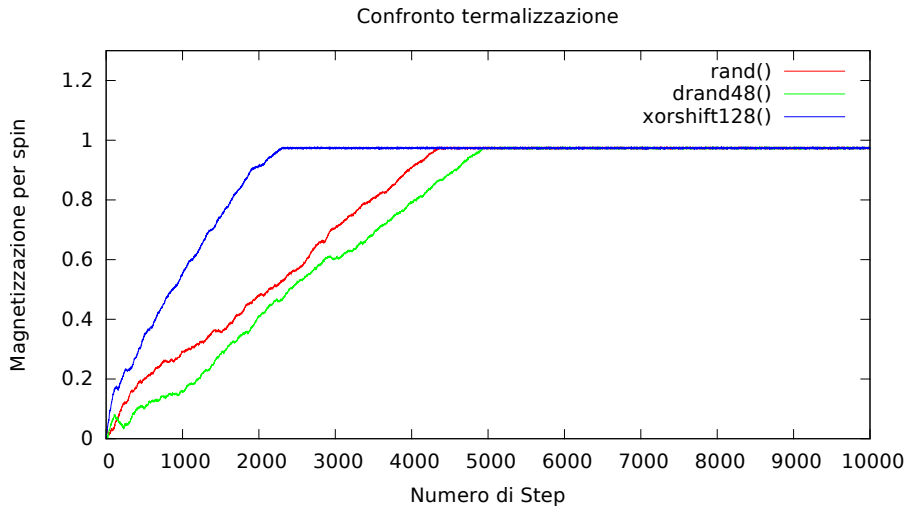

Figura 3: *Passi Montecarlo* necessari per raggiungere la termalizzazione in funzione a tre diversi PNRG.

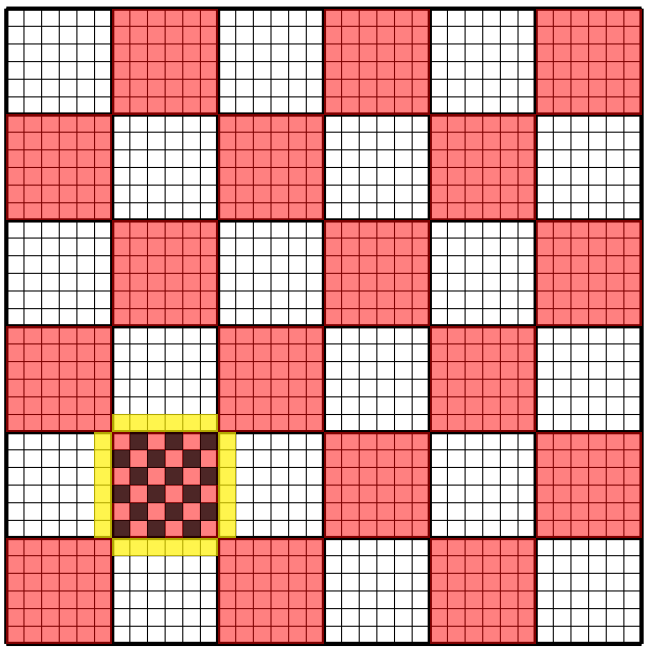

Figura 4: Strategia di update a doppia scacchiera. I blocchi vengono attivati a scacchiera. In ogni blocco, l'update degli spin è a sua volta fatto a scacchiera. In giallo sono identificati gli spin di *cornice* che devono essere storati nella memoria shared, di ogni blocco, per garantire la condizione di raccordo *blocco-blocco*

messi a disposizione. Oltre alla *memoria globale* è possibile usare un livello di memoria superiore chiamato *memoria shared*. Il vantaggio di questo tipo di memoria è che ha una velocità di lettura/scrittura superiore a quella globale. Lo svantaggio è che le sue dimensioni sono più piccole ed è suddivisa per blocco. Per far comunicare due dati, istanziati su blocchi diversi di memoria shared è necessario quindi scrivere delle condizioni di raccordo.

Un primo kernel naif che è possibile scrivere è quello che utilizza solo memoria globale e aggiorna gli spin seguendo solo una singola scacchiera. Un kernel più elaborato che sia in grado di gestire anche il livello di *memoria shared* segue una strategia di update dei siti esposta in Fig. 4.

Il kernel viene lanciato due volte con un parametro di offset. Al primo run vengono aggiornati tutti i blocchi rossi. A loro volta gli spin all'interno del blocco vengono aggiornati seguendo una strategia a scacchiera. È necessario organizzare anche una scacchiera tra i blocchi per non creare conflitto tra spin che rappresentano le condizioni di *raccordo blocco-blocco*, che in Fig. 4 sono rappresentati dalla *cornice gialla*.

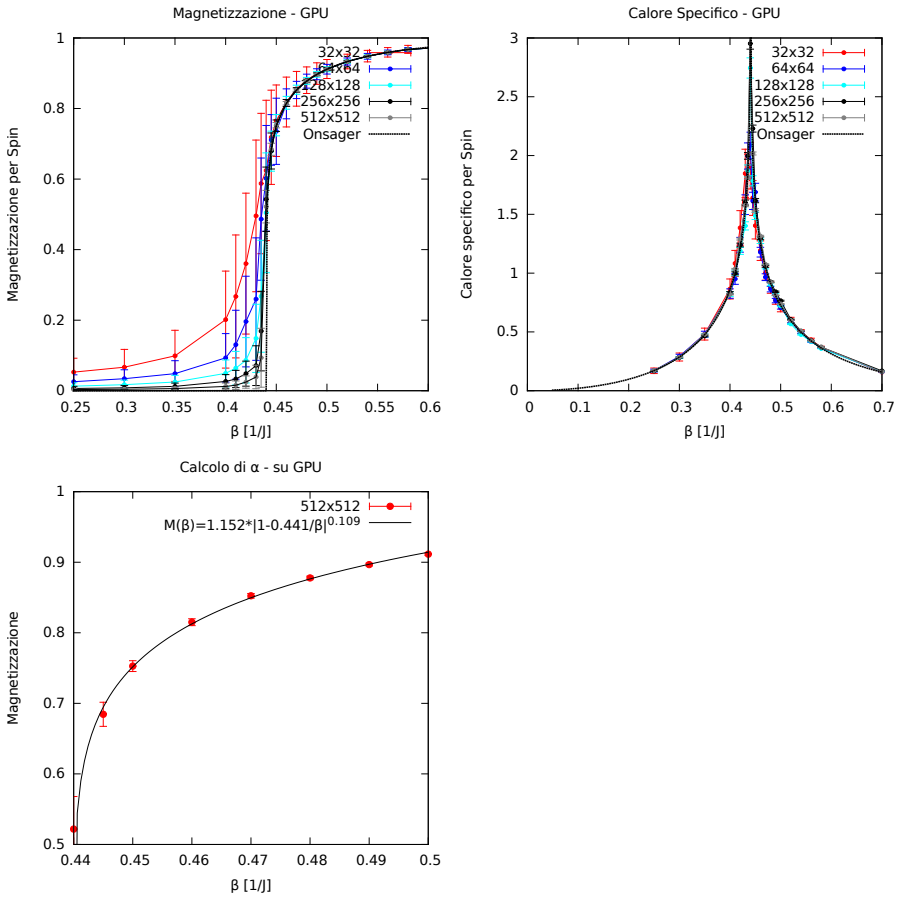

Figura 5: *Magnetizzazione* e *Calore Specifico* per diverse taglie di reticolo, in funzione di β generati su GPU

4.1 Consistenza del modello Fisico su GPU

In Fig. 2 e Fig. 5 vengono rappresentati tre grafici. Il primo rappresenta la magnetizzazione (per spin) in funzione di β, il secondo il calore specifico per spin sempre in funzione di β e nel terzo viene fatto un fit dei valori di magnetizzazione, per ricavare l'esponente critico α.

Nell'intorno del β_c la magnetizzazione si comporta come

$$M(\beta) \sim \left|1 - \frac{\beta_c}{\beta}\right|^{\alpha}. \tag{13}$$

Effettuando un fit dei valori intorno alla temperatura critica, lasciando come parametri liberi α e β_c è quindi possibile ricavare il valore dell'esponente critico e della temperatura critica di transizione di fase. In Fig. 2 e Fig. 5 sono riportati i valori del fit trovati che sono compatibili con quelli noti.

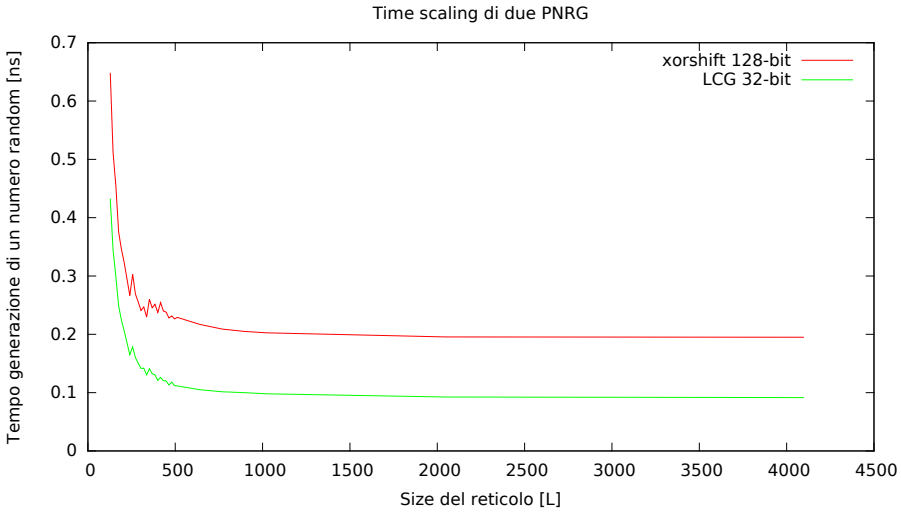

Figura 6: Confronto del tempo di generazione di un numero pseudorandom per due differenti PNRG

4.2 Confronto performance dei vari PNRG

Insieme al numero di step necessari a far termalizzare il sistema è necessario indagare anche le performance temporali di ogni singolo PNRG per verificare se l'effettivo guadagno in numero di step è un effettivo guadagno in termini di tempi di simulazione. Un generatore caratterizzato da una miglior statistica e randomicità dei dati può far termalizzare prima il sistema ma va comunque valutato il costo temporale necessario a generarli.

Mettendo insieme i risultati esposti in Fig. 3 ed in Fig. 6 si può notare che, l'algorimo *XorShift 128-bit* impiega circa 1600 step a termalizzare. Con la funzione *rand 31-bit* impega quasi 4000 step, quindi un generatore a $31bit$ impega circa 2.5 volte in più a termalizzare rispetto ad uno a 128 bit. Per quanto riguardo i tempi, un generatore a $128 - bit$, rispetto ad uno a $32 - bit$ è solamente 2 volte più lento. Questo fa sì che la soluzione più performante sia quella che prevede di utilizzare un generatore di numeri pseudorandom più lento ma che permetta al sistema di termalizzare in un numero inferiori di step Monte Carlo.

4.3 Compatibilità statistica dei risultati

In Tab. 1 sono riportati i valori di magnetizzazione per un reticolo 32×32, con relativo errore, in funzione di β. Come detto in precedenza si è riusciti a riportare completamente, operazione per operazione, l'algoritmo usato su CPU alle schede grafiche quindi ci si aspetta che dopo lo stesso numero di passi Monte Carlo, il dato sia proprio identicamente lo stesso.

Come si può notare in Tab. 1 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 l'evoluzione di uno spin è caratterizzato 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 caratterizava l'evoluzione dello spin nominato precedentemente, sulla GPU può caratterizzare un altro spin. È quindi corretto trovare valori differenti, ma compatibili entro il loro errore statistico.

In conclusione i dati ci indicano che nonostante ci sia un rimescolamento dei numeri casuali, la statistica del sistema rimane sempre la stessa quindi le due simulzioni su CPU e GPU restituiscono risultati compatibili tra loro. Entrambi i risultati hanno un andamento compatibile con quello atteso se confrontati col modello al limite termodinamico.

4.4 Ottimizzazione della dimensione del blocco

La dimensione del numero di threads appartenenti ad un blocco è un parametro che può essere scelto nel programma. In Fig. 7 viene presentato l'andamento del tempo di run in funzione della dimensione del blocco.

Come è possibile notare, il minimo è facilmente identificabile a $BLOCKL = 24$. Questo valore trova la sua spiegazione in come è costruita la scheda grafica. Per la GTX480, il numero di threads che possono essere contenuti in un blocco è al massimo 1024[4].

La spiegazione di questo risultato è nelle modalità di lancio del kernel. Per effettuare la doppia scacchiera viene passata una *dimBlock* sotto forma di struttura *dim*3 in cui il numero di threads in Y è la metà di quelli in X. Chiamando $BLOCKL$ la dimensione del blocco in Y, risulta che in un blocco, per ogni lancio del kernel ci sono un numero di threads che è pari a $2 * BLOCKL * BLOCKL$. Il valore di $BLOCKL$ che permette a questa espressione di saturare completamente il warp è appunto 24 in quanto $24 * 24 * 2 = 1152$. Quindi, per ottenere le migliori performance è opportuno scegliere una dimensione del blocco in modo da permettere la saturazione del Warp.

[4]Dato ottenuto con cudaGetDeviceProprieties()

β	$\langle M \rangle$-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.

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 8. 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 du 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.

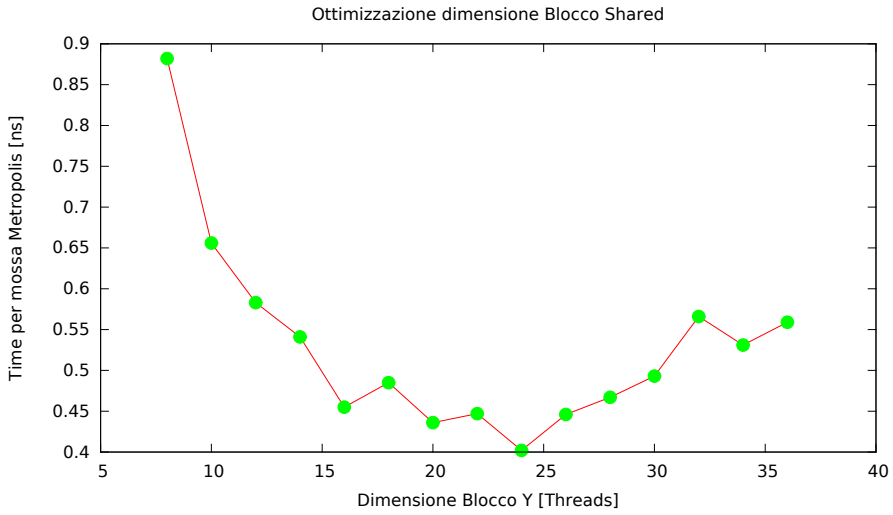

Figura 7

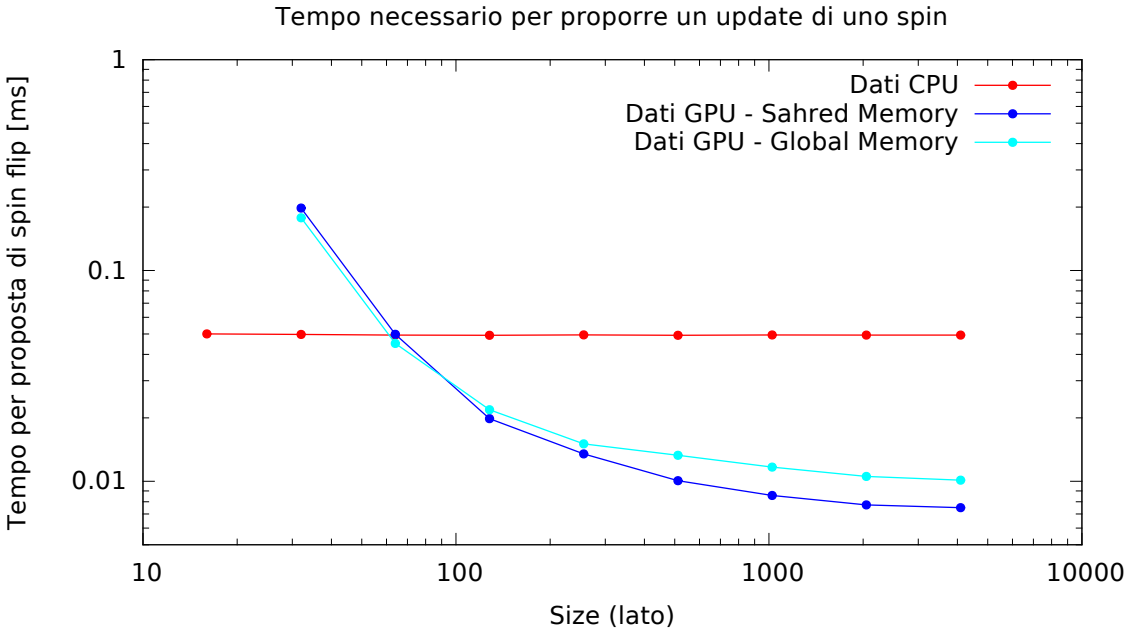

Figura 8: CPU/GPU