High Performance Computing
Prova scritta – 17 luglio 2019 – 1h30

PARTE 1 – RISPOSTA SINGOLA - Ogni domanda ha una sola risposta VERA.

- Una risposta esatta fa acquisire il punteggio positivo riportato a fianco della domanda
- Una risposta errata fa perdere il punteggio negativo riportato a fianco della domanda
- Una risposta lasciata in bianco viene valutata 0

1. **(2, -.5)** Con il termine *false sharing*
   a) Ci si riferisce ad una problematica di performance dovuta alla granularità del sistema di cache in un multicore
   b) Ci si riferisce ad una problematica di performance data dalla condivisione di risorse di calcolo in un multicore
   c) Ci si riferisce ad una problematica di correttezza funzionale dovuta alla granularità del sistema di cache in un multicore
   d) Ci si riferisce ad una problematica di correttezza funzionale dovuta alla condivisione di risorse di calcolo in un multicore

2. **(2, -.5)** Con il termine *strong scaling*
   a) Ci si riferisce alla capacità di un sistema di calcolo parallelo di raggiungere alti livelli di scalabilità di *speedup*
   b) Ci si riferisce alla capacità di un sistema di calcolo parallelo di ottenere *speedup* indipendentemente dalla dimensione del dataset
   c) Ci si riferisce ad un sistema di calcolo altamente parallelo (con un elevato numero di cores), in contrasto coi sistemi moderatamente paralleli, caratterizzati da *weak scaling*
   d) Nessuna delle precedenti risposte è vera

3. **(2, -.5)** La *condizione di Bernstein*
   a) Stabilisce che il numero di transistor in un circuito integrato CMOS raddoppia all'incirca ogni 18 mesi
   b) Stabilisce che con ogni nuova generazione di processo produttivo la percentuale di un chip che può operare a massima frequenza decade esponenzialmente per vincoli di potenza
   c) Stabilisce che il miglioramento che si può ottenere su una certa parte del sistema è limitato dalla frazione di tempo in cui tale attività ha luogo
   d) Stabilisce che due task T1 e T2 sono paralleli se l'input di T1 (T2) non è parte dell'output di T2 (T1) e se gli output di T1 e T2 non si sovrappongono

PARTE 2 – (POSSIBILI) RISPOSTE MULTIPLE -
Ogni domanda può avere da una a quattro risposte CORRETTE.

- Ogni risposta esatta viene calcolata: +1
- Ogni risposta errata viene calcolata: -0.5
- Una risposta lasciata in bianco viene calcolata: 0

4. Relativamente al sistema di memoria in CUDA
   a) Se tutti i *threads* di un *warp* eseguono una istruzione di *load* si ha *memory coalescing*
   b) Nella dichiarazione di variabili, la keyword `__device__` è opzionale, se usata con la keyword `__shared__` o con la keyword `__constant__`
   c) La shared memory è una *scratchpad* che contiene variabili esplicitamente gestite dal codice del *kernel*
   d) La tecnica del *tiling* non richiede sincronizzazione

5. Relativamente al sistema di memoria di una FPGA e della metodologia Vivado HLS
   a) Le problematiche di località dei dati tipiche dei sistemi CPU e GPU non sono importanti nel contesto FPGA
   b) Di default le variabili sono memorizzate all'interno di registri (blocchi di FF), mentre gli array sono memorizzati in BRAM
   c) Nella direttiva **`#pragma HLS INTERFACE m_axi port=dst offset=slave`**, il parametro offset=slave permette di gestire in maniera dinamica l'indirizzo nel quale andare a scrivere i dati in RAM
   d) I blocchi BRAM sono più veloci dei flip-flop, ma meno numerosi.

6. Il formato ELF
   a) Contiene sezioni – utilizzate durante le fasi di generazioni di un eseguibile – e segmenti – utilizzati durante il loading
   b) Contiene simboli che rappresentano variabili o funzioni, purché definiti nel file
   c) Contiene diverse sezioni per i dati, distinguendo tra quelli inizializzati e non inizializzati
   d) Può essere ispezionato tramite il tool *nlmconv* delle *binutils*

7. Il formato GIMPLE
   a) Non consente ottimizzazione del codice al di fuori della forma SSA
   b) Codifica sia il formato 3-address code che il CFG del programma
   c) Non consente operazioni con meno di due operandi
   d) È usato per l'implementazione del supporto OpenMP, ma prima che si entri nella forma SSA

PARTE 3 – DOMANDE APERTE

- Una risposta esatta fa acquisire il punteggio positivo riportato a fianco della domanda
- Una risposta errata può eventualmente causare una penalità che dipende dalla gravità dell'errore
- Una risposta lasciata in bianco viene calcolata: 0
- L'eventuale sforamento del limite di righe o parole (laddove imposto), porterà a una decurtazione di un punto per ogni riga
- SI RICORDA CHE L'UNICO FOGLIO DA CONSEGNARE E' IN CALCE AL COMPITO. QUESTO FOGLIO, PUO' SERVIRE ESCLUSIVAMENTE COME "BRUTTA COPIA". EVENTUALI RISPOSTE SCRITTE IN QUESTO FOGLIO NON VERRANNO PRESE IN CONSIDERAZIONE

8. **(9 pt)** Si descrivano le principali idee architetturali che differenziano una CPU tradizionale da una unità di calcolo GPU

   1. Rimozione dei componenti che tradizionalmente aiutano un singolo core a eseguire velocemente (esecuzione OoO, branch predictor, memory prefetcher, …)
   2. Fase di instruction fetch comune, ALU multiple: SIMD
   3. Favorire il calcolo inframezzato di numerosi frammenti di programma su un singolo core per nascondere l'alta latenza delle operazioni di memoria bloccanti

9. **(9 pt)** Si consideri un programma OpenMP con la seguente struttura:
   - una parte sequenziale: 50000 istruzioni;
   - un loop parallelo (#pragma omp parallel for): 10^4 iterazioni da 10000 istruzioni ciascuna;
   - un loop parallelo (#pragma omp parallel for): 10^2 iterazioni da 5000 istruzioni ciascuna;
   - una parte sequenziale: 10000 istruzioni.

   Assumendo che tutte le istruzioni eseguano in un singolo ciclo di CPU:

   a) Si calcoli lo speedup massimo ideale ottenibile (rispetto allo stesso programma eseguito in sequenziale) in assenza di overhead.

   b) Si calcoli lo speedup ottenibile su un sistema con 256 cores. Si assuma che (i) non venga specificata alcuna clausola schedule; (ii) l'overhead per creare una parallel region sia di 500 cicli; (iii) l'overhead generato dal chunking sia di 50 cicli in caso di static scheduling e di 100 cicli in caso di dynamic scheduling.

   c) Si calcoli il guadagno energetico[1] dell'esecuzione parallela rispetto a quella sequenziale, sullo stesso sistema di cui al punto b), assumendo che (i) ciascuno dei processori operi a 500 MHz e abbia un consumo di potenza pari a P_{dyn} = 0.1 W, P_{stat} = 0.02 W; (ii) il resto del sistema (cores esclusi) abbia un consumo di potenza pari a $P_{uncore} = P_{dyn\_uncore} + P_{stat\_uncore}$ = 0.08 W; (iii) i processori siano attivi durante la fase di attesa su una barriera (busy waiting).

[1] Si ricorda che l'energia spesa in un lasso di tempo t si può calcolare come $E = P * t$, dove P è il consumo di potenza istantaneo.

a) Tempo richiesto per l'esecuzione sequenziale del programma:

$$T_{seq} = 50000 + 10^4 * 10000 + 10^2 * 5000 + 10000 = 100.560.000 \text{ [cicli]}$$

Tempo richiesto per l'esecuzione parallela su p processori:

$$\boldsymbol{T_{par}} = 50000 + \frac{10^4 * 10000}{p} + \frac{10^2 * 5000}{p} + 10000 \text{ [cicli]}$$

Se applichiamo la legge di Amdahl, nel caso ideale per un numero di processori altissimo (tendente a infinito) le parti parallelizzabili eseguono in tempo trascurabile (tendente a zero). Di conseguenza il tempo richiesto per l'esecuzione parallela nel caso ideale è la somma delle parti sequenziali, ovvero 60.000 cicli.
Lo *speedup* ideale risultante è:

$$\boldsymbol{Speedup} = \frac{100.560.000}{60000} = \mathbf{1676}$$

Si potrebbe più precisamente osservare che le singole iterazioni dei loop non sono internamente parallelizzabili, e dunque parallelismo ideale corrisponde in realtà all'esecuzione di una sola iterazione di ciascun loop. In questo modo il tempo richiesto per l'esecuzione parallela diventa

60000+10000+5000

E lo speedup ideale diventa 100560000/75000=1340

b) Se consideriamo il tempo di esecuzione parallelo reale, occorre considerare gli overheads per il parallelismo. Due regioni parallele da 500 cicli ciascuna (il costo di creazione per una parallel region è fisso, indipendentemente dalla quantità di lavoro svolta dentro la regione stessa). Se non si specifica nessuna clausola di scheduling, come richiesto, il default è schedule (static), il che genera un breve pezzo di codice prima del loop che calcola lower e upper bound dei loop eseguiti dai vari threads. Quindi anche in questo caso l'overhead di chunking si paga una volta per loop, indipendentemente da quanto lavoro viene svolto nel loop.

Il tempo di esecuzione parallela su 256 cores diventa quindi

$$T_{par} = 50000 + \left\lceil \frac{10^4}{256} \right\rceil * 10000 + \left\lceil \frac{10^2}{256} \right\rceil * 5000 + 10000 + 500 * 2 + 50 * 2 = 466.100 \text{ [cicli]}$$

e lo speedup

$$\boldsymbol{Speedup} = \frac{100560000}{466100} = \mathbf{215.75}$$

c) Per calcolare l'energia occorrono i tempi in secondi:

$$T_{seq(s)} = \frac{T_{seq}}{f} = \frac{100.560.000}{500 * 10^6} = 0.20112 \text{ s}$$

Da cui possiamo ricavare l'energia dell'esecuzione sequenziale:

$$E_{seq} = \left[\left(P_{dyn} + P_{stat}\right) + (N-1) * P_{stat} + P_{uncore}\right] * T_{seq(s)} =$$
$$[(0.1 + 0.02) + 255 * 0.02 + 0.08] * 0.20112 = \mathbf{1.065936\ J}$$

Il tempo di esecuzione (in secondi) della sola parte sequenziale del calcolo parallelo è:

$$T_{seq\_part} = \frac{50000 + 10000}{500 * 10^6} = 0.00012 \text{ s}$$

Mentre il tempo di esecuzione (in secondi) della parte parallela è:

$$T_{par\_part} = \left(\left\lceil\frac{10^4}{256}\right\rceil * 10000 + \left\lceil\frac{10^2}{256}\right\rceil * 5000\right) * \frac{1}{500 * 10^6} = 0.00081\text{s}$$

L'energia della parte parallela è dunque:

$$E_{par} = \left[\left(P_{dyn} + P_{stat}\right) + (N-1) * P_{stat} + P_{uncore}\right] * T_{seq\_part} + \left[N * \left(P_{dyn} + P_{stat}\right) + P_{uncore}\right] * T_{par\_part} =$$
$$[(0.1 + 0.02) + 255 * 0.02 + 0.08] * 0.00012 + [256 * (0.1 + 0.02) + 0.08] * 0.00081 =$$
$$= \mathbf{0.025584\ J}$$

Il guadagno di energia è il rapporto tra questa grandezze:

$$EG = \frac{E_{seq}}{E_{par}} = \frac{1.065936}{0.025584} = \mathbf{41.66}$$