Nome e Cognome ____ Matricola __________________

Corso di *High Performance Computing*
Ingegneria e Scienze Informatiche—Università di Bologna

Demo esame

- La prova dura 60 minuti
- Durante la prova non è consentito consultare libri, appunti o altro materiale.
- Non è consentito l'uso di dispositivi elettronici (cellulari, tablet...), né interagire in alcun modo con gli altri studenti pena l'esclusione dalla prova, che potrà avvenire anche dopo il termine della stessa.
- Le risposte devono essere scritte **a penna** su questi fogli, in modo **leggibile**. Le parti illeggibili o scritte a matita saranno ignorate.
- Eventuali altri fogli possono essere utilizzati per la brutta copia ma non verranno valutati.
- I voti saranno pubblicati su AlmaEsami e vi verrà data comunicazione all'indirizzo istituzionale (@studio.unibo.it).
- I voti restano validi fino alla sessione d'esame di **settembre 2019** inclusa. Dopo tale data tutti i voti in sospeso saranno persi.

NON SCRIVERE NELLA TABELLA SOTTOSTANTE

D. 1	D. 2	D. 3	D. 4
/ 8	/ 8	/ 8	/ 8

Nome e Cognome ______________________________________ Matricola __________________

Domanda 1. Descrivere vantaggi e svantaggi delle architetture a memoria distribuita.

Commento sulle risposte date a questa domanda. Qualcuno ha menzionato il rapporto computazione/comunicazione (o viceversa) in modo scorretto. Non è vero, come sembra da alcune risposte, che le architetture a memoria distribuita "consentano" un elevato rapporto computazione/comunicazione. Al contrario, le architetture a memoria distribuita sono adeguate per quei problemi che esibiscono un elevato rapporto computazione/comunicazione, perché in tali architetture le comunicazioni sono (spesso) molto meno efficienti della computazione e vanno limitate al minimo. Detto in altri termini, l'elevato rapporto computazione/comunicazione deve essere una caratteristica del problema affinché sia possibile risolverlo su una architettura a memoria distribuita.

Nome e Cognome ____ Matricola ___________________

Domanda 2. Si consideri il seguente frammento di codice:

```
b[0] = f(a[0] + a[1]);
c[0] = 0;
d[0] = 0;
for (i = 1; i<n-1; i++) {
   b[i] = f( a[i-1] + a[i] + a[i+1] );
   c[i] = g( b[i-1] );
   d[i] = h( a[i-1] + b[i-1] );
}
b[n-1] = c[n-1] = d[n-1] = 0;
```

Assumendo che:

- l'array a[ ] sia stato inizializzato prima dell'inizio del ciclo;
- tutte le variabili e le funzioni siano definite in modo opportuno;
- le funzioni f() g() e h() restituiscano valori che dipendono solo dai rispettivi parametri;

ristrutturare il codice in modo che produca lo stesso risultato ma non siano presenti *loop-carried dependencies* nel ciclo "for".

Soluzione. Ci sono almeno due possibilità che consistono nell'applicare la tecnica di "loop aligning"; la prima è la seguente (le modifiche rispetto al ciclo originale sono evidenziate):

```
b[0] = f(a[0] + a[1]);
c[0] = 0;
d[0] = 0;
c[1] = g( b[0] );
d[1] = h( a[0] + b[0] );
for (i = 2; i<n-1; i++) {
   b[i-1] = f( a[i-2] + a[i-1] + a[i] );
   c[i] = g( b[i-1] );
   d[i] = h( a[i-1] + b[i-1] );
}
b[n-2] = f( a[n-3] + a[n-2] + a[n-1] );
b[n-1] = c[n-1] = d[n-1] = 0;
```

Si noti come sia necessario eseguire fuori dal ciclo il calcolo di c[1], d[1] e b[n - 2]. La seconda possibilità, del tutto equivalente, è:

```
b[0] = f(a[0] + a[1]);
c[0] = 0;
d[0] = 0;
c[1] = g( b[0] );
d[1] = h( a[0] + b[0] );
for (i = 1; i<n-2; i++) {
   b[i] = f( a[i-1] + a[i] + a[i+1] );
   c[i+1] = g( b[i] );
   d[i+1] = h( a[i] + b[i] );
}
b[n-2] = f( a[n-3] + a[n-2] + a[n-1] );
b[n-1] = c[n-1] = d[n-1] = 0;
```

Gli errori più comuni che ho visto in questo esercizio sono (*i*) la presenza di accessi out-of-bound nel caso in cui gli estremi del ciclo non siano stati settati correttamente; (*ii*) la mancata inizializzazione dei valori non calcolati all'interno del ciclo.

Nome e Cognome ______________________________________ Matricola __________________

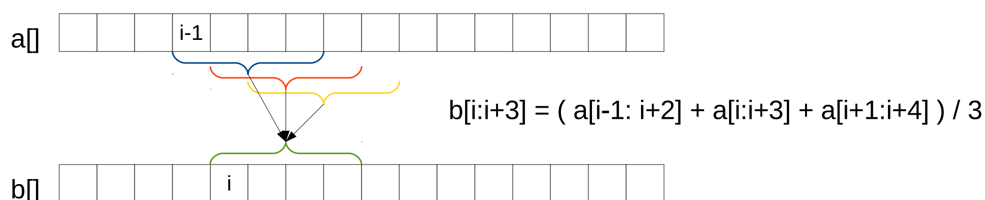
Domanda 3. Dopo aver scelto <u>due</u> tra i seguenti problemi:

1. Calcolo del prodotto scalare tra due array di valori reali;
2. Ricerca di k chiavi intere in un albero binario di ricerca con n nodi, $k << n$;
3. Individuare una chiave di cifratura mediante ricerca esaustiva (*brute-force*);
4. Dato un array a[N] di *float*, calcolare i valori di un nuovo array b[N] tale che b[i] = (a[i - 1] + a[i] + a[i + 1])/3 per ogni $1 \leq i \leq N - 2$.
5. Calcolo dell'insieme di Mandelbrot;

per ciascuno giustificate se ritenete più opportuno applicare il paradigma SIMD (*Single Instruction Multiple Data*) oppure MIMD (*Multiple Instruction Multiple Data*) per la risoluzione.

Commento sulle risposte date a questa domanda. Come ho spiegato a voce prima dell'inizio della prova, la parte più importante di questo esercizio è la spiegazione che veniva data; ciononostante, in alcuni casi la scelta era abbastanza ovvia

1. L'approccio preferibile è probabilmente quello SIMD: abbiamo visto in laboratorio che la versione SIMD del prodotto scalare è strettamente "imparentata" con la versione SIMD del calcolo della somma degli elementi di un array, e può essere realizzata in modo efficiente con poca fatica.
2. L'approccio preferibile è probabilmente quello MIMD, in quanto la ricerca di ciascuna chiave potrebbe avere esiti diversi (qualcuna delle chiavi potrebbe essere presente, qualcun'altra no), dando origine a computazioni divergenti che male si prestano al paradigma SIMD.
3. L'approccio preferibile è probabilmente quello MIMD che abbiamo applicato in laboratorio con OpenMP. L'approccio SIMD "naturale" consisterebbe nell'esaminare K chiavi contemporaneamente, essendo K l'ampiezza dei registri SIMD. Non è però detto che una chiave di cifratura possa essere interamente contenuta in un elemento di un registro SIMD (anzi, è abbastanza improbabile che ciò sia vero in generale). Non è nemmeno detto che l'algoritmo sia tale da poter essere "SIMD-izzato", cioè da poter cifrare/decifrare K chiavi contemporaneamente: se l'algoritmo in questione fa uso di istruzioni condizionali (if-then-else) o cicli con numero di iterazioni variabile in base alla chiave, mal si presterebbe all'implementazione SIMD. In generale non si può escludere a priori che un approccio SIMD sia possibile, ma chi ha scelto SIMD avrebbe dovuto almeno menzionare le questioni di cui sopra.
4. L'approccio preferibile è probabilmente quello SIMD: è infatti possibile calcolare K valori adiacenti in b[] usando lo schema:



   in cui si usano essenzialmente due somme SIMD e una divisione SIMD/scalare (oppure SIMD/SIMD nel caso in cui il compilatore trasformi lo scalare 3 in {3, 3, 3, 3}).
5. Qui molti hanno risposto "SIMD", dando come motivazione il fatto che le stesse operazioni sono applicate su tutti i pixel dell'immagine. Ciò non è completamente corretto, perché l'implementazione canonica dell'algoritmo di Mandelbrot vista a lezione contiene

computazioni divergenti, nel senso che ogni pixel può richiedere un numero di iterazioni diverso per divergere. Questo rende l'implementazione SIMD non ovvia (ma non impossibile, ad esempio https://github.com/skeeto/mandel-simd); però anche in questo caso chi ha optato per l'approccio SIMD avrebbe dovuto discutere la scelta in modo adeguato.

Nome e Cognome ______________________________________ Matricola __________________

Domanda 4. Commentare riga per riga il seguente programma CUDA, che compila ed esegue correttamente. Le righe da commentare sono quelle numerate (i numeri di riga NON fanno parte del programma). Che valore viene stampato alla riga 10?

```
        #include <stdio.h>

1       __global__ void my_kernel(int *x)
        {
2             *x = *x + 1;
        }

        int main(void)
        {
3             int h_a = 42;
4             int *d_a;
5             const size_t size = sizeof(int);
6             cudaMalloc((void **)&d_a, size);
7             cudaMemcpy(d_a, &h_a, size, cudaMemcpyHostToDevice);
8             my_kernel<<<1,1>>>(d_a);
9             cudaMemcpy(&h_a, d_a, size, cudaMemcpyDeviceToHost);
10            printf("%d\n", h_a);
11            cudaFree(d_a);
              return 0;
        }
```

Riga n.	Commento
1	
2	
3	
4	
5	
6	

Nome e Cognome ______________________________________ Matricola __________________

7	
8	
9	
10	
11	