Domanda 1. Si descriva un esempio di un problema che si ritiene particolarmente adatto ad essere risolto su una architettura a memoria distribuita, motivando la risposta (le risposte non motivate non saranno prese in considerazione).

Domanda 2. Si consideri il seguente ciclo (si assuma che tutte le variabili e le funzioni usate siano state definite in modo appropriato; in particolare, $f(x)$ restituisce un valore che dipende solo da x)

```
a[0] = f(0);
b[0] = 0;
c[0] = -3;
for (i=1; i<n; i++) {
   a[i] = f(i);
   b[i] = a[i-1] * 3;
   c[i] = (a[i-1] / 3) - b[i];
}
```

Ristrutturare il ciclo in modo che produca lo stesso risultato ma non siano presenti *loop-carried dependencies*.

Domanda 3. Descrivere l'architettura di una moderna GPU.

Domanda 4. Descrivere a parole una primitiva MPI di comunicazione collettiva; descrivere anche un possibile scenario d'uso, cioè un esempio concreto in cui può/deve essere usata.

Domanda 5. Descrivere vantaggi e svantaggi delle architetture a memoria distribuita.

Domanda 6. Dopo aver scelto due 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;

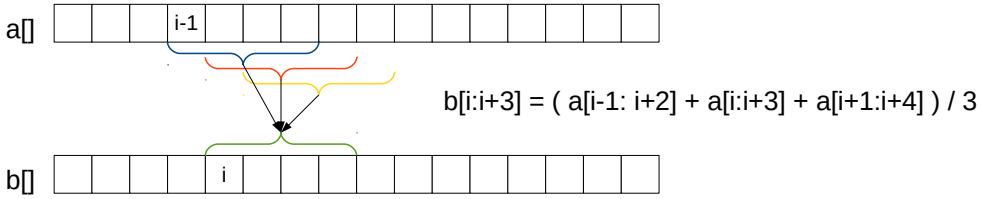
per ciascuno giustificate se ritenete più opportuno applicare il paradigma SIMD (*Single Instruction Multiple Data*) oppure MIMD (*Multiple Instruction Multiple Data*) per la risoluzione.

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.

Domanda 7. 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;
      }
```

Domanda 8. Descrivere i concetti di *speedup* ed *efficienza* di applicazioni parallele.

Domanda 9. Si consideri il seguente frammento di codice C che calcola le forze che agiscono su N particelle causate dalla mutua attrazione gravitazionale. La funzione gravity(i, j) (non dettagliata) calcola la forza che agisce tra le particelle i e j. La funzione è simmetrica, cioè gravity(i, j) = gravity(j, i), per cui basta calcolare le forze che agiscono tra le particelle i, j con $i < j$. Si assuma che il valore restituito dalla funzione gravity(i, j) dipenda solo da i e j, e da nessun'altra proprietà delle particelle (in particolare, non dipende dal valore del campo `force`).

```
struct Particle {
      float force; /* la forza è rappresentata da un singolo valore reale */
      /* altri attributi della particella, non mostrati */
};

Particle particles[N];
int i, j;

...

for (i=0; i<N; i++) {
      for (j=i+1; j<N; j++) {
            const float f = gravity(i, j);
            particles[i].force += f;
            particles[j].force += f;
      }
}
```

Discutere i problemi e le possibili soluzioni legate all'uso di OpenMP per parallelizzare il codice precedente.

Domanda 10. Scegliere due delle seguenti primitive MPI di comunicazione collettiva:

- **`MPI_Scatter`**
- **`MPI_Scan`**
- **`MPI_Reduce`**

Per ciascuna delle primitive scelte:

1. Si descriva in modo sintetico ma il più possibile esauriente che cosa fa (non è necessario ricordarsi la sintassi); se lo si desidera si può accompagnare la spiegazione con schemi e diagrammi.
2. Si illustri un caso concreto in cui tale primitiva può essere impiegata; è possibile ricorrere ad esempi visti in aula e/o in laboratorio.

Domanda 11. Si illustrino i concetti di *thread* e *thread block* delle architetture CUDA.