Matricola: _____

Nome: _____

Cognome: _____

ESAME
Programmazione Avanzata e Parallela

12 febbraio 2024

L'esame consiste di 10 domande a risposta multipla sugli argomenti del corso. Ogni domanda può ricevere un punteggio massimo di *tre* punti. Affinché una risposta sia considerata valida la scelta *deve essere motivata*. Una risposta errata o non motivata riceverà *zero* punti.

Domanda 1

Si supponga di avere il seguente `Makefile`:

```
CC = gcc
CFLAGS = -O3 -Wall --std=c11 --pedantic

all: program

program: main.o bar.o foo.o
	$(CC) $^ -o $@

main.o: main.c
	$(CC) $(CFLAGS) -c $<

%.o: %.c %.h
	$(CC) $(CFLAGS) -c $<

.PHONY: clean

clean:
	rm *.o
```

Si supponga che sia stato invocato `make` e successivamente il file `foo.c` sia modificato. Alla successiva invocazione di `make` quali comandi saranno eseguiti?

☐ gcc -O3 -Wall -c foo.c

☒ gcc -O3 -Wall main.o bar.o foo.o -o program

☐ $(CC) $(CFLAGS) -c foo.c

☐ gcc -O3 -Wall -c foo.c

gcc -O3 -Wall -c bar.c bar.h

gcc -O3 -Wall -c foo.c foo.h

☐ gcc -O3 -Wall -c main.c

gcc -O3 -Wall main.o bar.o foo.o -o program

---

prima dovremo aggiornare foo.o e poi di conseguenza program dunque 1) gcc -03 -Wall -c foo.c ed a seguire gcc -03 -Wall main.o bar.o foo.o foo.o -o program

---

Domanda 2

Si supponga di avere questi due frammenti di codice C:

Frammento 1

```
for (int i = 0; i < n; i++) {
    if (v[i] < 10) {
        s += v[i];
    } else {
        s += 10;
    }
}
```

Frammento 2

```
for (int i = 0; i < n; i++) {
    int q = (v[i] < 10);
    s += q * v[i] + (1 - q) * 10;
}
```

e si assuma di avere variabili `s`, `v` definite e del tipo corretto.
Quale delle seguenti affermazioni è corretta?

☐ I due codici non sono equivalenti

☐ Il frammento 2 è sempre eseguito in minor tempo del frammento 1

☐ Il frammento 1 è sempre eseguito in minor tempo del frammento 2

☒ Quale sia il frammento più veloce dipende dalla distribuzione dei valori in `v`

Domanda 3

Sia dato il seguente frammento di codice C:

```
for (int i = 0; i < n; i++) {
  v[i] = 0;
  for (int j = 0; j < n; j++) {
    v[i] += q[j * n + i];
  }
}
```

Questo non è vero: se fossimo in un architettura a 64 bit per esempi

dove `v` è un vettore di n elementi e `q` è un vettore di n^2 elementi rappresentate una matrice quadrata di lato n in ordine *row-major*.
Quale delle seguenti affermazioni è corretta?

- ☐ Per questa tipologia di accesso una rappresentazione di `q` come lista concatenata con gli elementi nello stesso ordine di `q` avrebbe garantito maggiore località di memoria
- ☒ L'accesso sarebbe più efficiente se `q` fosse in forma *column-major*
- ☐ Il ciclo più interno può essere correttamente parallelizzato con OpenMP aggiungendo solamente `#pragma omp parallel for`
- ☐ L'accesso sarebbe più efficiente se anche `c` fosse in forma *row-major*

Poiché stiamo accedendo a valori appartenenti alla stessa colonna una forma column-major sarebbe più efficente perché sfrutterebbe la località spaziale

---

Domanda 4

Dato il seguente codice che utilizza due diverse rappresentazioni per gli stessi dati (i.e., due strutture differenti):

```
#include <stdio.h>
#include <stdlib.h>

struct S1 {
  uint32_t a1;
  uint64_t a2;
  uint32_t a3;
};

struct S2 {
  uint32_t a1;
  uint32_t a3;
  uint64_t a2;
};
```

Quale delle seguenti affermazioni è *errata*?

☐ Le due strutture possono avere dimensioni differenti

☐ Il numero di strutture in una linea di cache può essere diverso tra `S1` e `S2`

☒ Le due strutture avranno sempre lo stesso padding

☐ Le strutture potrebbero avere padding alla fine, non solo tra i diversi campi

---

Questo non è vero: se fossimo in un architettura a 64 bit per esempio s1 occuperebbe 3 blocchi di memoria mentre s2 solo 2 blocchi di memoria

---

Domanda 5

Quale delle seguenti affermazioni sull'I/O è *errata*?

☐ `fread` può leggere meno del numero di elementi richiesti

☒ `mmap` inserisce `EOF` per indicare dove terminano i dati

☐ `fread` può essere più efficiente di chiamate consecutive a `getc`

☐ Se dobbiamo svolgere un accesso non sequenziale ai dati contenuti nel file possiamo utilizzare `fseek`

---

non è vero che mmap inserisce EOF per indicare dove terminano i dati, bensì "mappa" una determinata quantità di memoria multipla delle dimensioni delle pagine (poiché mmap lavora con la memoria virtuale)

Domanda 6

Dato il seguente codice C facente uso di OpenMP:

```c
#include <stdio.h>
#include <omp.h>

float * random_matrix(int n, int m) { /* ... */ }

int main(int argc, int * argv[])
{
  const int n = 1000;
  float * M = random_matrix(n, n);
  float s = 0;
#pragma omp parallel
  {
    float ps = 0;
#pragma omp for collapse(2)
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        ps += M[i * n + j];
      }
    }
#pragma omp critical
    {
      s += ps;
    }
  }
  printf("%f\n", s);
  return 0;
}
```

Si supponga che la funzione `random_matrix` sia correttamente definita e ritorni un vettore di valori floating point tra 0 e 1.
Quale delle seguenti affermazioni è corretta?

- ☐ Vi è una race condition dato che l'accesso a `M` non è in una sezione critica
- ☒ Sarebbe stato possibile aggiungere `nowait` a `#pragma omp for collapse(2)` senza modificare la correttezza del codice
- ☐ Per essere corretto `#pragma omp critical` dovrebbe essere sostituito da `#pragma omp single`
- ☐ La variabile `ps` dovrebbe essere definita prima di `#pragma omp parallel`

Poiché l'operazione a seguire è in un critical, e dunque solo un thread alla volta può eseguirlo, non importa l'ordine con cui i thread sommino la propria somma parziale alla somma totale e dunque il nowait potrebbe essere usato senza problemi

---

Domanda 7

Dato il seguente codice Python:

```
class A:

    def g(self, x):
        self.f(x + 1)

    def f(self, x):
        print(f"A {x}")


class B(A):

    def g(self, x):
        self.f(x + 2)


class C(B):

    def f(self, x):
        print(f"C {x}")


class D(C):

    def g(self, x):
        super().g(x + 1)


x = D()
x.g(1)
```

quale è il valore risultante dall'ultima riga (i.e., `x.g(1)`)?

☐ `A 2`

☐ `C 3`

☒ `C 4`

☐ Viene generata una eccezione perché tutte le classi devono implementare sia `g` che `f`

D : g(1) -> call at super so -> C : g(2) -> call at super so -> B: g(4) = f(4) -> call at C again -> f(4) : print("C 4")

Domanda 8

Dato il seguente codice Python:

```
def f(g, x):
    def h1(y):
        return x + 2*y

    def h2(z):
        return g(h1(x))

    return h2


func = f(lambda x: x + 2, 6)
```

Quale è il valore ritornato da `func(3)`:

☐ Non è possibile usare lambda come argomenti di funzioni

☒ 20

☐ 18

☐ 8

---

f(x+2, 6) -> g = x+2, x = 6 so h1(y) = 6 + 2*y and then h2(z) = g(h1(6)) so it become h1(6) = 6 + 2*6 = 18 -> g(18) = 18 +2 = 20

Domanda 9

Dato il seguente codice Python:

```
class StrangeException(Exception):
    pass


class UnusualException(Exception):
    pass


def f(x):
    try:
        g(x)
    except Exception:
        print("A")
    except StrangeException:
        print("B")


def g(x):
    try:
        if (x < 0):
            raise StrangeException()
        if (x > 10):
            raise UnusualException()
    except UnusualException:
        print("C")


f(-3)
f(30)
```

Cosa viene stampato a schermo al termine dell'esecuzione?

☒ `A C`

☐ `C C`

☐ `B C`

☐ Viene stampato un messaggio di errore perché l'eccezione `StrangeException` non è catturata

---

Nel primo caso viene alzata Strange Exception, che non viene catturata in g quindi arriva ad f che la cattura come semplice Exception e printa A. Nel secondo caso invece viene alzata Unusual Exception che viene direttamente catturata in g e printa C

Domanda 10

Dato il seguente codice Python:

```
def f(x):
    while True:
        yield lambda y: x + y
        x += 1


h = f(0)
for i in range(5):
    g = next(h)
    print(g(i))
```

Quali sono i valori stampati a schermo?

☐ `<function f.<locals>.<lambda> at 0x104eef7f0>`

☐ `<function f.<locals>.<lambda> at 0x104eef760>`
`...`

☐ `0 1 2 3 4`

☐ Viene generata una eccezione dato che non è possibile chiamare `g`

☒ `0 2 4 6 8`

---

f ha come x iniziale 0, restituisce una lambda del tipo lambda y : y + 0, quindi g(0) = 0 + 0 = 0 poi x = 1 e dunque g(1) = 2; poi x = 2 e g(2) = 4 and so on

---
