Matricola: _____

Nome: _____

Cognome: _____

ESAME
Programmazione Avanzata e Parallela

10 luglio 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:

```
all: main

main: A.o B.o C.o D.o
	gcc $^ -o main

%.o: %.c
	gcc -c $< -o $@
```

Se, dopo aver precedentemente invocato `make` viene modificato il file `A.c`, alla successiva invocazione di `make` quali comandi saranno eseguiti

☒
```
gcc -c A.c -o A.o
gcc A.o B.o C.o D.o -o main
```

☐
```
gcc -c A.c -o A.o
gcc -c B.c -o B.o
gcc -c C.c -o C.o
gcc -c D.c -o Do
gcc A.o B.o C.o D.o -o main
```

☐
```
gcc -c A.c -o A.o
```

☐
```
gcc A.o B.o C.o D.o -o main
```

---

servirà innanzitutto aggiornare A.o e dopodiché ricorsivamente si risolverà la dipendenza di main da A.o dunque in ordine gcc -c A.c -o A-o e poi gcc A.o B.o C.o D.o -main

---

Domanda 2

Si supponga di avere questo frammento di codice C:
Frammento 1

```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    for (int k = 0; k < n; k++) {
      s += v[???]
    }
  }
}
```

e si assuma di avere variabili `s`, `v` definite e del tipo corretto. La variabile `v` punta a un vettore di n^3 elementi.
Quale delle seguenti espressioni da sostituire a `???` ci aspettiamo porti le migliori prestazioni?

☒ `i*n*n + j*n + k`

☐ `k*n*n + i*n + j`

☐ `i*n*n + k*n + j`

☐ `k*n*n + j*n + i`

Essendo un vettore in memoria è allocato in maniera contigua. Se io quindi accedo ogni volta a v[k*n*n + ...] ad ogni passo farò salti di dimensione n*n, il che non sfrutta la località spaziale. Questa viene sfruttata invece accedendo a v[i*n*n + ...]. Qui di nuovo devo scegliere tra un salto di dimensione n (con v[i*n*n + k*n + j]) o un salto di dimensione 1 (con v[i*n*n + j*n + k]) e dunque scelgo la seconda opzione

---

Domanda 3

Date le tre seguenti strutture dati:

A Lista concatenata

B Array

C Lista concatenata "srotolata" in cui ogni nodo contiene fino a $k > 1$ elementi

si mettano in ordine crescente rispetto a quanto traggono beneficio dall'uso della cache nel caso sia necessario scorrere *in ordine* gli elementi in esse contenuti.
Nota: l'ordine è dalla struttura dati che tra maggior beneficio a quella che ne trae di meno.

☐ A B C

☐ A C B

☒ B C A

☐ C B A

L'Array sfrutta sempre al massimo la località di memoria per costruzione (non deve mai sprecare spazio con eventuali puntatori a nodi successivi). La lista concatenata srotolata è meglio di una lista concatenata normale in quanto sfrutta di più la località di memoria ma poiché deve tenere i puntatori ai nodi successivi è meno efficiente dell'array.

Domanda 4

Le operazioni di prefetching esplicite (usando e.g., intrinsics fornite dal compilatore) posso aiutare a mitigare gli effetti di:

- ☐ Branch non predetti correttamente
- ☒ L'accesso a indirizzi di memoria il cui valore potrebbe non essere in cache
- ☐ Gli structural hazard
- ☐ Accessi a memoria che potrebbero generare race conditions

---

Tramite il builtin prefetch possiamo effettivamente indicare al compilatore della memoria da caricare in anticipo in cache poiché ci aspettiamo che verrà usata.

Domanda 5

Quale delle seguenti affermazioni sulle operazioni di input/output è corretta?

- ☐ Se usiamo `fseek` per muoverci alla fine del file dobbiamo necessariamente leggere tutto il contenuto del file
- ☐ `fread` richiede come argomento un buffer che deve avere come dimensione una potenza di due
- ☒ `fseek` può avere un offset negativo
- ☐ `getc` e `putc` lavorano solamente su file che sono stati mappati in memoria con `mmap`

Questo è vero poiché in alcuni casi potremmo dover tornare indietro dal punto in cui siamo e dunque potremmo dover eseguire un seek relativo alla posizione corrente in negativo

---

Domanda 6

Dato il seguente codice C facente uso di OpenMP:

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

int * random_int_vector(int n) {  /* ... */  }

int main(int argc, int * argv[])
{
  const int n = 1000;
  int * v = random_int_vector(n);
  int count8 = 0;
#pragma omp parallel
  {
    int c = 0;
#pragma omp for nowait
    for (int i = 0; i < n; i++) {
      c += (v[i] == 8);
    }
    /* TODO */
  }
  printf("%d\n", count8);
  return 0;
}
```

Si supponga che la funzione `random_int_vector` sia correttamente definita.
Al posto del commento `/* TODO */` cosa è necessario inserire perché il codice funzioni correttamente?

☐ `count8 += c;`

☐ nulla

☐
```
#pragma omp single
count8 += c;
```

☒
```
#pragma omp critical
count8 += c;
```

Se vogliamo sommare in count8 le somme parziali presenti in c avremo bisogno che tramite una sessione critica (eseguita con critical) ogni thread separatamente acceda a count8 e ci sommi il proprio valore locale della somma parziale c

---

Domanda 7

Dato il seguente codice Python:

```
class A:

    def f(self):
        print("A")


class B(A):

    def f(self):
        print("B")


class C(A):
    pass


class D(A):

    def g(self, x):
        super().f()
        x.f()


x = D()
y = B()
z = C()
x.g(y)
x.g(z)
```

cosa viene stampato a schermo?

☐ `A B A B`

☒ `A B A A`

☐ `<class '__main__.A'> B <class '__main__.A'> A`

☐ `A B A C`

Chiamato x.g(y) per prima cosa si invoca f definito in A, e dunque print A. poi f definito in B (poiché y è istanza di B) e dunque print B. poi abbiamo x.g(z), dunque di nuovo f definito in A, e di nuovo print A, e poi f definito in C (poiché z è istanza di C) ma in tal caso in C non è ridefinita f, la quale dunque viene presa da A (super classe di C) e dunque di nuovo print A. A B A A

Domanda 8

Dato il seguente codice Python:

```
def f(n):
    def g(x):
        return lambda y: 3*x + 2*y
    xs = []
    for i in range(n):
        xs.append(g(i))
    return xs


res = []
for func in f(5):
    res.append(func(3))
```

Quale è il valore finale di `res`?

☐ Il codice non è sintatticamente corretto

☐ `[<function f.<locals>.g at 0x1012f3490,<function f.<locals>.g at 0x1012f3490, ...]`

☒ `[6, 9, 12, 15, 18]`

☐ `[9, 11, 13, 15, 17]`

---

f ritorna una lista del tipo : [lambda y:2*y, lambda y : 3 + 2*y ... lambda y : 3*(n-1) + 2*y]
dunque ne deriva che in res otteniamo 6, 3 + 6, 6 + 6, 9 + 6, 12 +6

Domanda 9

Dato il seguente codice Python:

```
def deco(n):
    def g(func):
        def f(*args):
            x = func(*args)
            return x**n
        return f
    return g


@deco(3)
def h(x, y):
    return x + 2*y + 1
```

Quale è il valore ritornato da `h(2,1)`?

☐ Il codice ritorna una closure

☒ `125`

☐ `5`

☐ Il codice non è sintatticamente corretto

Abbiamo che g prende h come parametro dunque poi f calcola h(2, 1) ed ottiene 5. Dopodiché entra in gioco il decoratore che prende g ed eleva il valore alla n, in questo caso 3 ed otteniamo 5^3 ergo 125

Domanda 10

Dato il seguente codice Python:

```
def g(n):
    for i in range(2, n):
        if i % 7 == 0:
            yield i


for i in g(40):
    print(i)
```

Quali sono i valori stampati a schermo?

☐ `<generator object g at 0x104907bc0>` ripetuto 40 volte

☐ Il codice genera una eccezione in quanto non è chiamato `next` sull'iteratore

☒ `7 14 21 28 35`

☐ `2 3 4 5 ... 39`

---

g(40) genera effettivamente la successione 7, 14, 21, 28, 35 e dunque poi i itera su questa

---
