

## Programmazione Funzionale e Parallela (A.A. 2015-2016)

Corso di Laurea in Ingegneria Informatica e Automatica  
Sapienza Università di Roma

A

**Esame del 14/12/2015 – Durata 1h 30' (solo esonerati)**

Inserire nome, cognome e matricola nel file `studente.txt`.

---

### Esercizio 1 (Ricerca delle occorrenze di una stringa in un testo mediante OpenCL)

Lo scopo dell'esercizio è quello di scrivere una funzione C che cerca tutte le occorrenze di una stringa in un testo in parallelo usando OpenCL. Si vada nella directory di lavoro `finds` e si definisca nel file `finds.c` la funzione `finds` con il seguente prototipo:

```
void finds(const char* pattern, const char* text, char* out,
           int n, clut_device* dev, double* t)
```

dove:

- `pattern`: stringa da cercare;
- `text`: array di `n` caratteri (non terminato con `'\0'`) che rappresenta il testo in cui cercare la stringa;
- `out`: vettore di output di dimensione `n` (si veda sotto come calcolarlo);
- `n`: numero di caratteri del testo;
- `dev`: ambiente di esecuzione della GPU (si veda `clut.h`);
- `t`: parametro in cui restituire la durata dell'esecuzione del kernel.

La funzione deve inizializzare `out` in modo che, per ogni `i` in  $[0, n-1]$ , `out[i]` vale 1 se la stringa `pattern` compare in `text` a partire dall'indice `i`, e 0 altrimenti. Scrivere un opportuno kernel OpenCL nel file `finds.cl`. Suggerimento: usare un `NDRange` monodimensionale.

Per compilare usare il comando `make`. Per effettuare dei test usare `make test1` e `make test2`.

---

### Esercizio 2 (Decrittazione mediante vettorizzazione)

Si vuole vettorizzare la seguente funzione di decrittazione utilizzando intrinsic SSE o AVX:

```
void decode(const char* key, int m, char* str) {
    int i, n = strlen(str);
    for (i=0; i<n; i++) str[i] -= key[i % m];
}
```

La funzione prende una stringa `str` di lunghezza `n` e un array `key` di `m` `char` e la decodifica sottraendo al codice del carattere di indice `i` il valore `key[i % m]`. La soluzione deve avere costo  $O(n/m)$ , assumendo che ogni operazione vettoriale richieda tempo  $O(1)$ . Si può assumere che `m` non sia superiore a 16.

Scrivere la soluzione nel file `decode.c` nella directory di lavoro `decode`. Si usi il `main` di prova fornito come test.

---

### Esercizio 3 (Filtri grafici mediante multi-threading)

Lo scopo dell'esercizio è quello di scrivere un modulo C basato su `pthread` che, data in input un'immagine a 256 toni di grigio di dimensione  $w \times h$ , crei una nuova immagine ottenuta da quella di input mediante sfocatura, come nell'esempio sotto.



(a) Immagine originale



(b) Immagine dopo il blur

L'effetto è ottenuto definendo il tono di grigio del pixel  $(i,j)$  nell'immagine di output come la media aritmetica dei toni di grigio dei pixel dell'immagine di input nella finestra  $7 \times 7$  centrata nel pixel  $(i,j)$ . I valori dei pixel della finestra che escono dai bordi dell'immagine devono essere ignorati ai fini del calcolo della media.

Si vada nella directory di lavoro `blur` e si definisca nel file `blur.c` la funzione `blur_mt` con il seguente prototipo:

```
void blur_mt(const unsigned char* A,  
             unsigned char* B, size_t w, size_t h)
```

dove:

- A: puntatore a un buffer di dimensione `w*h*sizeof(unsigned char)` byte che contiene l'immagine di input in formato row-major<sup>1</sup>;
- B: puntatore a un buffer di dimensione `w*h*sizeof(unsigned char)` byte che contiene l'immagine di output in formato row-major;
- w: larghezza di `in` in pixel (numero di colonne della matrice di pixel);
- h: altezza di `in` in pixel (numero di righe della matrice di pixel);

La funzione deve creare **almeno due thread** per effettuare l'applicazione del filtro in parallelo su più core. Per compilare usare il comando `make`. Per effettuare un test usare `make test`. Verrà prodotta l'immagine di output `colosseo-blurred.pgm`.

---

<sup>1</sup> Cioè con le righe disposte consecutivamente in memoria.