Esercitazione 02

17 ottobre 2023

Lo scopo di questa esercitazione è quello di implementare il merge di due vettori **ordinati** rispettivamente di n_1 e n_2 elementi in un terzo vettore di $n_1 + n_2$ elementi. Questa è una fase essenziali di algoritmi come mergesort. Si ricorda che questa operazione di merge può essere effettuata in tempo lineare.

Il file `main.c`, il file header `merge.h` e un makefile sono forniti, deve essere scritto il file `merge.c`, del quale è fornito solo uno scheletro.

In esso devono essere scritte due implementazioni distinte dell'operazione di merge di vettori ordinati, una facente uso di `if ... else` e l'altra senza di essi nel ciclo principale. Come indicato nel file `merge.h`, le due funzioni saranno della forma:

```
void merge(int * v1, int n1,
           int * v2, int n2,
           int * results);

void merge_branchless(int * v1, int n1,
                      int * v2, int n2,
                      int * results);
```

Dove `v1` e `v2` sono i due vettori di input rispettivamente di lunghezza `n1` e `n2` e `results` è un vettore di lunghezza `n1 + n2` che conterrà il risultato del merge di `v1` e `v2` (il suo contenuto sarà quindi sovrascritto).

Una volta implementate le due versioni di merge è necessario mostrare i tempi medi di esecuzione (almeno 10 ripetizioni) che hanno su vettori casuali di dimensione compresa tra 1000 e 20000 (in incrementi di 1000). Si vedano le note per le funzioni di test già implementate ed utilizzabili per questo passo.

Si ricorda che la procedura di merge ha la seguente struttura:

```
1. Siano v1 e v2 due vettori di input di lunghezza n1 e n2
2. Sia r il vettore di output di lunhezza n1+n2
3. Iniziamo da i=0 e j=0
4. finché i < n1 e j < n2
   4.1. Se v1[i] < v2[j] inseriamo v1[i] in r e incrementiamo i
   4.2. Altrimenti inseriamo v2[j] in r e incrementiamo j
5. finiamo di copiare gli elementi mancanti di v1 o v2 in r
```

Note

1. Per evitare la generazione di istruzioni predicative (predicate instructions) come CMOV (x86-64) o CSEL (ARM64) si consiglia di usare gcc con i flag `-fno-if-conversion -fno-if-conversion2 -fno-tree-loop-if-convert`. Questi flag sono comunque già presenti

nel makefile fornito. Normalmente questi flag non devono essere attivati perché inibiscono delle ottimizzazioni che spesso vogliamo ci siano, qui sono utilizzati solo per mostrare la differenza tra codice con e senza branch.

2. Viene già fornita una funzione `test_merge` che ritorna il tempo di esecuzione di una operazione di merge. Come argomenti prende la dimensione da testare `n` e la funzione `m` da chiamare per l'operazione di merge. Viene anche fornito `avg_test_merge` che richiama `test_merge` un numero di volte indicato da `repetitions` e ritorna il tempo medio in millisecondi speso per ogni chiamata.

3. Viene fornita anche una funzione `check_merge` che compie il merge di due vettori di 5 elementi e stampa il risultato. Può essere utile per verificare la correttezza della propria implementazione.

4. Il file `main.c` è modificabile per testare meglio l'implementazione di `merge` e `merge_branchless` (e.g., verificarne la correttezza chiamando `check_merge`). Il file `merge.h` non deve essere modificato.

Extra

Una volta completata la scrittura della procedura di merge è possibile scrivere mergesort. Si scriva un file `mergesort.c` con una funzione `main` e una funzione `mergesort` di signature:

```
merge_sort(int * v,
           int len,
           void (* m) (int *, int, int *, int, int *))
```

Questa funzione deve implementare mergesort in modo iterativo (non ricorsivo).

Si modifichino le funzioni `test_merge`, `avg_test_merge` e `check_merge` affinché testino `merge_sort` e si aggiuna al Makefile un target `mergesort`. Si compari la versione con branch con quella branchless del mergesort.