Sistemi di Calcolo (A.A. 2015-2016)
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

D

Esame del 20/1/2016 (non esonerati) – Durata 1h 30'

Esercizio 1 (gerarchie di memoria)

Si consideri la seguente funzione che opera su un vettore di `n` interi:

```
int vec_sum(short* v, int n) {
    int sum = 0;
    while (n-- > 0) sum += v[n];
    return sum;
}
```

Assumere che: 1) le variabili del programma siano tutte tenute in registri della CPU; 2) le funzioni siano eseguite su un sistema con una cache da 128 KB avente linee da 128 byte; 3) l'array sia allineato a un indirizzo multiplo di 32; 4) `n` = 1000000; 4) `sizeof(short)==2`.

1. Quanti accessi a memoria vengono effettuati da ciascuna versione?
2. Quanti cache miss vengono generati? Illustrare il procedimento usato per il calcolo.
3. La dimensione della linea conta? E quella della cache?

Inserire le risposte nel file `es1.txt`.

Esercizio 2 (costrutti di ciclo in assembly x86)

Si traduca in assembly IA32 la seguente funzione C scrivendo un modulo `es2.s`:

```
void count_space(char* s, int* c) {
    *c = 0;
    while (*s) {
        if (*s == 32) (*c)++;
        s++;
    }
}
```

Per i test, usare il seguente programma di prova `es2-main.c`:

```
#include <stdio.h>
void count_space(char* s, int* c);
int main() {
    char s1[]="Star Wars 7 ";
    char s2[]="Obi-Wan Kenobi";
    char s3[]=" Chewbecca";
    char s4[]="";
    int c1, c2, c3, c4;
    count_space(s1, &c1);
    printf("%d [corretto = 3]\n", c1);
    count_space(s2, &c2);
    printf("%d [corretto = 1]\n", c2);
    count_space(s3, &c3);
    printf("%d [corretto = 1]\n", c3);
    count_space(s4, &c4);
    printf("%d [corretto = 0]\n", c4);
    return 0;
}
```

Generare un file eseguibile `es2` compilato con `gcc –m32`.

Esercizio 3 (chiamate a funzione in assembly x86)

Si traduca in assembly IA32 la seguente funzione C scrivendo un modulo `es3.s`:

```
int df(int x, int y);
int fun(int* x, int* y) {
    if (x>y) return df(*x*2,*y);
    return 7*df(*x,*y);
}
```

Per i test, usare il seguente programma di prova `es3-main.c`:

```
#include <stdio.h>
int fun(int* x, int* y);
int main() {
    int x, y, res;
    x = -1, y = 2, res = fun(&x, &y);
    printf("fun(%d,%d)=%d [corretto=-21]\n", x, y, res);
    x = 5, y = 1, res = fun(&x, &y);
    printf("fun(%d,%d)=%d  [corretto=28]\n", x, y, res);
    x = 7, y = 9, res = fun(&x, &y);
    printf("fun(%d,%d)=%d  [corretto=-14]\n", x, y, res);
    return 0;
}
```

E il seguente modulo `es3-util.s`:

```
.globl df
df: movl 4(%esp), %eax
    subl 8(%esp), %eax
    movl $0xABADCAFE, %ecx
    movl $0xCAFEBABE, %edx
    ret
```

Generare un file eseguibile `es3` compilato con `gcc –m32`.

Esercizio 4 (profilazione e ottimizzazione di programmi)

Si crei nel file `es4-opt.c` una versione ottimizzata del seguente modulo `es4.c`:

```
#include "es4.h"

// conta punti inclusi nella circonferenza passante per l'origine
// centrata nel punto p
int count(int* x, int* y, int p, int n) {
    int i, c;
    for (i = c = 0; i < n; i++)
        if (distsqr(x[p], y[p], x[i], y[i]) <
            distsqr(x[p], y[p], 0, 0)) c++;
    return c;
}

// conta il massimo numero di punti che sono inclusi nella
// circonferenza passante per l'origine centrata in uno dei
// punti dell'insieme
int count_max(int* x, int* y, int n) {
    int i, max = 0;
    for (i=0; i<n; i++)
        if (count(x, y, i, n) > max)
            max = count(x, y, i, n);
    return max;
}
```

Compilare due versioni del programma, usando `gcc` **a 32 bit** con livello di ottimizzazione 1 e lo stesso modulo `es4-main.c`:

1. Non ottimizzata: eseguibile `es4`.
2. Ottimizzata: eseguibile `es4-opt`.

Ai fini dell'ottimizzazione:

1. Usare `gprof` per identificare le porzioni più onerose computazionalmente. Chiamare gli eseguibili usati per la profilazione `es4-pg` e `es4-opt-pg`. Salvare i report di `gprof` nei file `es4.txt` e `es4-opt.txt`, rispettivamente
2. Esaminare il modulo assembly `es4.s` generato a partire da `es4.c` con `gcc –S -O1` per capire quali ottimizzazioni siano già state effettuate dal compilatore.

Rispondere alle seguenti domande:

1. Descrivere le ottimizzazioni applicate e dire perché si ritiene che siano efficaci.
2. Riportare il tempo di esecuzione di `es4` e di `es4-opt` usando il comando `time`.
3. Riportare i flat profile delle due versioni usando `gprof`.
4. Di quante volte è più veloce l'eseguibile `es4-opt` rispetto a `es4`?
5. Usando i dati del profilo `es4.txt`, calcolare lo speedup massimo che si può ottenere ottimizzando la funzione `count`.

Inserire le risposte nel file `es4.txt`. Alla fine del compito, **non eliminare i seguenti file**:

- `es4`
- `es4-pg`
- `es4.txt`
- `es4-opt`
- `es4-opt-pg`
- `es4-opt.txt`
- `gmon.out`