

2º curso / 2º cuatr.

Grado en
Ing. Informática

Arquitectura de Computadores: Exámenes y Controles

Examen de Prácticas 04/09/2012 resuelto

Material elaborado por los profesores responsables de la asignatura:
Mancia Anguita, Julio OrtegaLicencia Creative Commons 

1 Enunciado Examen de Prácticas del 04/09/2012

Cuestión 1.(1 punto)

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int n, ck, i, nth, ith, b[32];
    if(argc < 4) { printf("[ERROR]-Faltan parámetros\n"); exit(-1); }
    n = atoi(argv[1]); ck = atoi(argv[2]); nth = atoi(argv[3]);
    if(n>32) n=32; omp_set_num_threads(nth);
    #pragma omp parallel if(n>8) private(ith)
    { int a = 0; ith=omp_get_thread_num();
      #pragma omp single copyprivate(a)
      { printf("\nIntroduce valor de inicialización a: "); scanf("%d", &a);
        printf("\nSingle ejecutada por el thread %d\n", ith);
      }
      #pragma omp for schedule(static,ck)
      for (i=0; i<n; i++) b[i] = ith + a
    }
    for (i=0; i<n; i++) printf(b[%d]=%d, "i, b[i]);
    printf("\n");
}
```

(a) (0.2)

(b) (0.4)

(c) (0.2)



(d) (0.2)

Cuestión 2. (1 punto)

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int i, n, a[n], suma=0, sumap=0;
    if (argc < 2) {    fprintf(stderr, "\nFalta iteraciones\n");    exit(-1);    }
    n = atoi(argv[1]); if (n>20) n=20;
    for (i=0; i<n; i++)    a[i] = i;
    #pragma omp parallel
    {
        #pragma omp for schedule(static,1)
        for (i=0; i<n; i++)    sumap+= a[i];
        #pragma omp atomic
        suma += sumap;
    }
    printf("Fuera de 'parallel' suma=%d sumap=%d\n", suma, sumap); return(0);
}
```

(a) (0.2)

(b) (0.3)

>

>

(c) (0.2)

(d) (0.3)

2 Solución Examen de Prácticas del 04/09/2012

Cuestión 1.(1 punto)



```

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
(1) int n, ck, i, nth, ith, b[32];
(2) if(argc < 4) { printf("[ERROR]-Faltan parámetros\n"); exit(-1); }
(3) n = atoi(argv[1]); ck = atoi(argv[2]); nth = atoi(argv[3]);
(4) if(n>32) n=32; omp_set_num_threads(nth);
(5) #pragma omp parallel if(n>8) private(ith)
(6) { int a = 0; ith=omp_get_thread_num();
(7) #pragma omp single copyprivate(a)
(8) { printf("\nIntroduce valor de inicialización a: "); scanf("%d", &a);
(9) printf("\nSingle ejecutada por el thread %d\n", ith);
(10) }
(11) #pragma omp for schedule(static,ck)
(12) for (i=0; i<n; i++) b[i] = ith + a
(13) }
(14) for (i=0; i<n; i++) printf(b[%d]=%d, "i, b[i]);
(15) printf("\n");
}

```

(a) (0.2)

(b) (0.4)

(c) (0.2)

(d) (0.2)

Solución

(a)

(11) (12)

(9)-(10)

(12)

(b)

(2)

(8) (9)

(14) (15)



- ❖ El primero (línea (2)) se ejecuta cuando el número de parámetros o argumentos en el comando que ejecuta el programa es menor que tres. En la ejecución de este apartado hay tres parámetros, luego no se ejecuta este primer `printf`.
- ❖ Los dos `printf` de la construcción `single` (líneas (8) (9)) los ejecutará sólo un thread de los que ejecutan la región `parallel`, el primero que llegue a la región `single`. El primer `printf` imprime, tras un salto de línea “\n”, el siguiente aviso al usuario “Introduce valor de inicialización a:”. El segundo (“\nSingle ejecutada por el thread %d\n”, `ith`) imprime un salto de línea, el texto “Single ejecutada por el thread”, el identificador del thread, `ith`, que ejecuta el código de la construcción `single` y otro salto de línea.
- ❖ Los dos `printf` que hay después de la construcción `parallel` (líneas (14) (15)) los ejecuta el thread 0, el master. El segundo `printf` imprimen un salto de línea. El primero está dentro de un `for` que imprime el contenido del vector `b` que se ha inicializado en paralelo dentro de la región `parallel`. El valor que se imprime para `b[i]` (`=a+ith`) va a depender del thread, `ith`, que ejecute la iteración `i` del bucle. El código que hay dentro de la construcción `parallel` fuera de la construcción `single` lo ejecutará además del thread 0 todos los threads que se creen por la directiva `parallel`. Según la cláusula `schedule(static,ck)` de la directiva `for`, la asignación de las iteraciones del bucle `for` de la construcción `parallel` se realiza con una planificación estática (`static`) que reparte unidades de trabajo de `ck` iteraciones consecutivas del bucle en turno rotatorio (`round-robin`) entre los `nth` threads. Dado el código “`n = atoi(argv[1]); ck = atoi(argv[2]); nth = atoi(argv[3]);`” (línea (3)) tendríamos que :
 - El primero de los argumentos, `argv[1]`, se almacena en la variable `n`, que se usa como número de componentes del vector `b` que se van a utilizar en el código y es el número de iteraciones de los dos bucles, en particular, del `for` de la construcción `parallel`.
 - El segundo de los argumentos, `argv[2]`, se almacena en la variable `ck`, que se usa como el número de iteraciones del bucle que contienen las unidades de código que se van a usar en la asignación de trabajo a los threads. Si `ck` no divide a `n` (número de iteraciones del bucle `for`) entonces habrá un trozo con menos de `ck` iteraciones.
 - El tercero de los argumentos, `argv[3]`, se almacena en la variable `nth`, que se usa como número de threads que van a ejecutar la región paralela del código.

Teniendo en cuenta el comando utilizado para ejecutar el programa, `examen1sep 20 2 4`, los valores de `n`, `ck` y `nth` son 20, 2 y 4 respectivamente; por tanto:

- Como `n=20` es mayor que 8 (cláusula `if` de la directiva `parallel`) la región paralela la ejecutan los 4 threads (`nth=4`) especificados en el último parámetro de entrada.
- Las 20 iteraciones del bucle se dividen en unidades de trabajo de 2 (`ck=2`) iteraciones cada una.
- Como hay 20 iteraciones (`n=20`), se tienen $20/2 = 10$ unidades.

La asignación a los 4 threads de las 20 iteraciones del bucle y, por tanto, las asignaciones de las componentes de `b` a threads es la mostrada en la Figura 1.

<code>ith =</code>	0	1	2	3	0	1	2	3	0	1
<code>i =</code>	0 1	2 3	4 5	6 7	8 9	10 11	12 13	14 15	16 17	18 19

Figura 1. Reparto de las iteraciones (`i=0,1,...19`) del bucle en unidades de dos iteraciones y asignación de unidades a los `nth=4` threads (`ith=0,1,2 y 3`)



Según esta asignación el penúltimo `printf` del código (línea (14)) imprimirá lo siguiente si, por ejemplo, `a` es 0:

`b[0]=0, b[1]=0, b[2]=1, b[3]=1, b[4]=2, b[5]=2, b[6]=3, b[7]=3, b[8]=0, b[9]=0, b[10]=1, b[11]=1, b[12]=2, b[13]=2, b[14]=3, b[15]=3, b[16]=0, b[17]=0, b[18]=1, b[19]=1.`

Para `a=10` imprimirá:

`b[0]=10, b[1]=10, b[2]=11, b[3]=11, b[4]=12, b[5]=12, b[6]=13, b[7]=13, b[8]=10, b[9]=10, b[10]=11, b[11]=11, b[12]=12, b[13]=12, b[14]=13, b[15]=13, b[16]=10, b[17]=10, b[18]=11, b[19]=11.`

(c) En este caso se aplica lo comentado en el apartado anterior (b) excepto en lo que respecto a los valores concretos que imprime el penúltimo `printf` (el que imprime los componentes del vector `b`).

Teniendo en cuenta el comando utilizado para ejecutar el programa, `examen1sep 8 2 4`, los valores de `n`, `ck` y `nth` son 8, 2 y 4 respectivamente:

- Como `n=8` NO es mayor que 8, debido a la cláusula `if` de la directiva `parallel` (línea (5)), la región paralela la ejecutará sólo el thread 0, el *master*.

Todas las iteraciones se asignarán, por tanto, al thread 0. Según esta asignación el penúltimo `printf` del código imprime lo siguiente si, por ejemplo, `a` es igual a 10:

`b[0]=10, b[1]=10, b[2]=10, b[3]=10, b[4]=10, b[5]=10, b[6]=10, b[7]=10, b[8]=10, b[9]=10, b[10]=10, b[11]=10, b[12]=10, b[13]=10, b[14]=10, b[15]=10, b[16]=10, b[17]=10, b[18]=10, b[19]=10.`

(d) En este caso también se aplica lo comentado en el apartado anterior (b) excepto en lo que respecto a los valores concretos que imprime el penúltimo `printf` (el que imprime los componentes del vector `b`).

Teniendo en cuenta el comando utilizado para ejecutar el programa, `examen1sep 13 4 3`, los valores de `n`, `ck` y `nth` son 13, 4 y 3 respectivamente, por tanto:

- Como `n=13` es mayor que 8 (cláusula `if` de la directiva `parallel`) la región paralela la ejecutan los 3 threads (`nth=3`) fijados con el último parámetro.
- Las 13 iteraciones del bucle se dividen en unidades de trabajo de 4 (`ck=4`) iteraciones cada una, excepto una de las unidades que tendrá menos iteraciones debido a que 4 no divide a 13.
- Como hay 13 iteraciones (`n=13`), se tienen $13/4 = 3$ unidades de 4 iteraciones y 1 unidad de 1 ($=13 \bmod 4$) iteración.

La asignación a los 3 threads de las 13 iteraciones del bucle y, por tanto, las asignaciones de los componentes de `b` a threads es la mostrada en la Figura 2.

<code>ith =</code>	0	1	2	0
<code>i =</code>	0 1 2 3	4 5 6 7	8 9 10 11	12

Figura 2. Reparto de las iteraciones (`i=0,1,...,12`) del bucle en unidades de cuatro iteraciones y asignación de unidades a los `nth=3` threads (`ith=0,1,2`)

Según esta asignación el penúltimo `printf` del código imprime lo siguiente si, por ejemplo, `a` es 10:

`b[0]=10, b[1]=10, b[2]=10, b[3]=10, b[4]=11, b[5]=11, b[6]=11, b[7]=11, b[8]=12, b[9]=12, b[10]=12, b[11]=12, b[12]=10.`

**Cuestión 2.(1 punto)**

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int i, n, a[n], suma=0, sumap=0;
    if (argc < 2) {    fprintf(stderr, "\nFalta iteraciones\n");    exit(-1); }
    n = atoi(argv[1]); if (n>20) n=20;
    for (i=0; i<n; i++)    a[i] = i;
    #pragma omp parallel
    {
        #pragma omp for schedule(static,1)
        for (i=0; i<n; i++)    sumap+= a[i];
        #pragma omp atomic
        suma += sumap;
    }
    printf("Fuera de 'parallel' suma=%d sumap=%d\n",suma, sumap); return(0);
}
```

(a) (0.2)

(b) (0.3)

>

>

(c) (0.2)

(d) (0.3)

Solución

(a)



(b)

>

>

>

(c)

```

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int i, n=20, a[n], suma=0, sumap=0, sumap2=0;
    if (argc < 2) {
        fprintf(stderr, "\nFalta iteraciones\n"); exit(-1);
    }
    n = atoi(argv[1]); if (n % 2) n+=1; if (n>20) n=20;
    for (i=0; i<n; i++) a[i] = i;
    #pragma omp parallel firstprivate(sumap, sumap2)
    {
        #pragma omp for schedule(static,1)
        for (i=0; i<n; i+=2) {
            sumap+= a[i]; sumap2+= a[i+1];
        }
        sumap += sumap2;
        #pragma omp atomic
        suma += sumap;
    }
    printf("Fuera de 'parallel' suma=%d sumap=%d\n",suma, sumap); return(0);
}

```



(d)

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int i, n=20, a[n], suma=0, sumap=0, sumap2=0;
    if (argc < 2) {
        fprintf(stderr, "\nFalta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n % 2) n+=1; if (n>20) n=20;
    for (i=0; i<n; i++) a[i] = i;
    #pragma omp parallel
    {
        #pragma omp for schedule(static,1) reduction(+:sumap,sumap2)
        for (i=0; i<n; i+=2) {
            sumap+= a[i];
            sumap2+= a[i+1];
        }
    }
    suma = sumap+sumap2;
    printf("Fuera de 'parallel' suma=%d sumap=%d\n",suma, sumap); return(0);
}
```

