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 Ortega

1 Enunciado Examen de Prácticas del 04/09/2012

Cuestión 1.(1 punto) Conteste a las siguientes cuestiones sobre el código `examen1sep.c` de la siguiente figura (considere que la variable de control `dyn_var` está a `false`):

```
#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) Indique qué hace el código y la orden que usaría para compilar desde una ventana de comandos (*Shell* o intérprete de comandos) si el ejecutable se quiere llamar `examen1sep` (utilice en la compilación alguna de las opciones de optimización que ha utilizado en la práctica de optimización de código).

(b) (0.4) Razone qué `printf` de los que hay en el código se ejecutan, qué threads ejecutan cada uno de ellos y qué imprime el programa en cada uno de estos `printf` si el usuario ejecuta: `examen1sep 20 2 4`.

(c) (0.2) Razone qué imprime el programa en cada uno de los `printf` si el usuario ejecuta: `examen1sep 8 2 4`.

(d) (0.2) Razone qué imprime el programa en cada uno de los `printf` si el usuario ejecuta: `examen1sep 13 4 3`.

Cuestión 2. (1 punto) Conteste a las siguientes cuestiones sobre el código `examen2sep.c` de la siguiente figura (considere que la variable de control `dyn_var` está a `false`):

```
#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) Razone qué imprime el programa en el último `printf` si el usuario ejecuta `examen2sep 20`.

(b) (0.3) Añada lo necesario al código (sin eliminar nada) para que al imprimir la variable `suma` se obtenga la suma de los `n` componentes del vector `a`. Razone las modificaciones realizadas. Razone qué valor numérico se obtiene en la pantalla cuando se imprimen `suma` y `sumap` si se ejecuta en las aulas de prácticas en una ventana de comandos lo siguiente:

```
> export OMP_NUM_THREADS=2
> examen2sep 10
```

(`examen2sep` se ha generado con el código fuente de este apartado (b))

(c) (0.2) Aplique desenrollado de bucles al código resultante del apartado (b). Escriba el código con las modificaciones realizadas.

(d) (0.3) Si se usa la cláusula `reduction` en la versión de código resultante del apartado (c), ¿qué eliminaría del código y qué añadiría al código y por qué? Escriba el código con las modificaciones descritas.

2 Solución Examen de Prácticas del 04/09/2012

Cuestión 1.(1 punto) Conteste a las siguientes cuestiones sobre el código `examen1sep.c` de la siguiente figura (considere que la variable de control `dyn_var` está a `false`):

```
#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) Indique qué hace el código y la orden que usaría para compilar desde una ventana de comandos (*Shell* o intérprete de comandos) si el ejecutable se quiere llamar `examen1sep` (utilice en la compilación alguna de las opciones de optimización que ha utilizado en la práctica de optimización de código).

(b) (0.4) Razone qué `printf` de los que hay en el código se ejecutan, qué threads ejecutan cada uno de ellos y qué imprime el programa en cada uno de estos `printf` si el usuario ejecuta: `examen1sep 20 2 4`.

(c) (0.2) Razone qué imprime el programa en cada uno de los `printf` si el usuario ejecuta: `examen1sep 8 2 4`.

(d) (0.2) Razone qué imprime el programa en cada uno de los `printf` si el usuario ejecuta: `examen1sep 13 4 3`.

Solución

(a) El código inicializa en paralelo los `n` componentes de un vector `b` (líneas (11) (12)) a partir de un valor `a` que se pide al usuario que introduzca (líneas (9)-(10)) y del identificador de los `nth` threads (`ith=0,1,…nth-1`) que ejecutan en paralelo la inicialización. La componente `i` de `b` se inicializa con `a+ith` (líneas (12)). Por tanto, el valor de `b[i]` va a depender del thread que ejecute la iteración `i` del bucle. Debido a la cláusula `if(n>8)` que acompaña a la directiva `parallel`, si n es menor o igual que 8 inicializa b sólo un thread, el máster (en este caso `ith`=0 para todos los componentes de `b`). Los valores de `b` tras la inicialización se imprimen en pantalla.

Para compilar se usaría: `gcc -O2 -fopenmp -o examen1sep examen1sep.c`

(b) Hay un total de 5 `printf` en el código, uno antes de la construcción `parallel` (línea (2)), dos en la construcción `parallel` (líneas (8) (9)), que están dentro de la construcción single, y dos después de la construcción `parallel` (líneas (14) (15)). Para la ejecución `examen1sep 20 2 4` se ejecutan todos los `printf` excepto el primero:

- 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.

ith =	0	1	2	3	0	1	2	3	0	1
i =	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)) imprimiría 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 imprimiría:

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 mod 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.

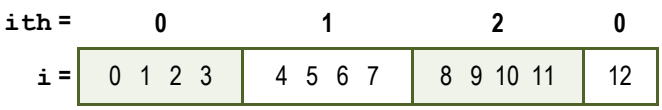

Figura 2. Reparto de las iteraciones (`i`=0,1,…12) del bucle en unidades de cuatro iteraciones y asignación de unidades a los `nth`=3 threads (`ith`=0,1,2)

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) Conteste a las siguientes cuestiones sobre el código `examen2sep.c` de la siguiente figura (considere que la variable de control `dyn_var` está a `false`):

```
#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) Razone qué imprime el programa en el último `printf` si el usuario ejecuta `examen2sep 20`.

(b) (0.3) Añada lo necesario al código (sin eliminar nada) para que al imprimir la variable `suma` se obtenga la suma de los `n` componentes del vector `a`. Razone las modificaciones realizadas. Razone qué valor numérico se obtiene en la pantalla cuando se imprimen `suma` y `sumap` si se ejecuta en las aulas de prácticas en una ventana de comandos lo siguiente:

```
> export OMP_NUM_THREADS=2
> examen2sep 10
```

(`examen2sep` se ha generado con el código fuente de este apartado (b))

(c) (0.2) Aplique desenrollado de bucles al código resultante del apartado (b). Escriba el código con las modificaciones realizadas.

(d) (0.3) Si se usa la cláusula `reduction` en la versión de código resultante del apartado (c), ¿qué eliminaría del código y qué añadiría al código y por qué? Escriba el código con las modificaciones descritas.

Solución

(a) El código imprime la variable compartida `suma` que contendrá un valor igual al número de thread que ejecutan la región paralela multiplicado por el valor que tenga la variable compartida `sumap` al terminar la ejecución paralela del bucle `for` de la región paralela. La directiva `for` añade una barrera implícita. También imprime la variable compartida `sumap`, su valor no es determinístico; es decir, la ejecución del código varias veces con los mismos parámetros de entrada no imprime siempre lo mismo. Esto ocurre porque los threads acceden a `sumap` en paralelo y todos la leen, modifican y escriben (R-M-W): `sumap+=a[i]`. Si los threads accedieran secuencialmente (es decir, un thread detrás de otro) a

`sumap+=a[i]` `(R-M-W)` al imprimir `sumap` se imprimiría la suma de todos los componentes del vector `a`.

(b) Para que, al imprimir `suma`, se obtenga la suma de `n` los componentes de `a`, la variable `sumap` debe ser privada a los threads y debe estar inicializada a 0 en todos los threads. Para conseguirlo, sin eliminar código, podemos usar una cláusula `firstprivate(sumap)` en la directiva `parallel`. Con esta opción se sustituiría el código "`#pragma omp parallel`" por este otro "`#pragma omp parallel firstprivate(sumap)`". Con `firstprivate` además de crear una variable privada `sumap` en todos los threads estas variables privadas se inicializan con el valor que tiene la variable global `sumap`, que en este caso es 0.

Si se ejecuta en una ventana de comandos:

```
> export OMP_NUM_THREADS=2
> examen2sep 10
```

se imprime lo siguiente:

```
> Fuera de 'parallel' suma=45 sumap=0
```

porque:

1. Al imprimir `suma` se imprime un valor que coincide con la suma de los componentes del vector `a`. Las 10 componentes de `a` se inicializa de forma que `a[i]=i`; por tanto, la suma de todas ellas será igual a 10*(10-1)/2=45.
2. Al imprimir `sumap` se imprime 0, que es el valor al que se inicializó la variable compartida `sumap`.

(c) Se va a aplicar, por ejemplo, un desenrollado de 2 iteraciones del bucle, es decir, en cada iteración del nuevo bucle se va a realizar dos iteraciones del bucle original:

```
#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) Cambios a realizar en el código:

- Se elimina la cláusula `firstprivate(sumap, sumap2)` de la directiva `parallel` y se añade a la directiva `for` una cláusula de reducción con el operador "+" y una lista de dos variables sumap y sumap2: `reduction(+:sumap,sumap2)`.
- Ya no es necesario usar una directiva `atomic` para que cada thread añada la suma parcial que han calculado a la variable compartida `suma`. Se eliminaría, por tanto, el siguiente código:

```
sumap += sumap2;
#pragma omp atomic
    suma += sumap;
```

- Se debe añadir código tras la construcción `parallel` que sume los contenidos de todas las variables compartidas que aparecen en la lista de la cláusula `reduction`, en este caso, son dos. Se añadiría después de la construcción `parallel`: `suma = sumap+sumap2;`

El código resultante sería el siguiente:

```
#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);
}
```