Documento anónimo

AC_Examen_Prac_2013_07_05_resuelto.pdf

Exámenes Resueltos (teoría y Prácticas)

2º Arquitectura de Computadores

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
UGR - Universidad de Granada

2º curso / 2º cuatr.

Grado en Ing. Informática

Doble Grado en Ing. Informática y Matemáticas

Arquitectura de Computadores: Exámenes y Controles

Examen de Prácticas AC 05/07/2013 resuelto

Material elaborado por los profesores responsables de la asignatura:
Mancia Anguita, Julio Ortega

1 Enunciado Examen de Prácticas del 05/07/2013

Cuestión 1.(0.5 puntos) **(a)** (0.05) ¿Cuántos nodos de cómputo (servidores) tiene atcgrid y cuántos chips de procesamiento (encapsulados, procesador) tiene una placa madre de atcgrid? **(b)** (0.05) ¿Cuántos cores de procesamiento tiene un chip de procesamiento (encapsulado) de atcgrid? **(c)** (0.05) ¿Qué gestor de colas ha utilizado en prácticas para enviar trabajos a atcgrid? **(d)** (0.25) Suponga que debe ejecutar en atcgrid el fichero ejecutable `hello` que tiene en su home del PC del aula de prácticas (o en su PC portátil si está trabajando en el aula de prácticas con su propio portátil), ¿qué haría para ejecutarlo en atcgrid? ¿Cómo sabría que ya ha terminado la ejecución? ¿dónde podría consultar los resultados de la ejecución? **(e)** (0.1) Suponga que debe ejecutar en atcgrid el *script* `scripthello.sh` que tiene en su home del PC del aula de prácticas (o en su PC portátil si está trabajando en el aula de prácticas con su propio portátil), ¿qué haría para ejecutarlo?

Cuestión 2. (1.2 puntos) Conteste a las siguientes cuestiones sobre el código `examen1.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>
int main(int argc, char **argv)
{  int i, n=20, tid, chunck;
   int a[n],suma=0,sumalocal;
   if(argc < 3)    {
      fprintf(stderr,"[ERROR]- Faltan datos de entrada \n");    exit(-1);
    }
   n = atoi(argv[1]); if (n>20) n=20; chunck = atoi(argv[2]);
   for (i=0; i<n; i++)    a[i] = i;
   #pragma omp parallel num_threads(4) default(none) private(sumalocal,tid) shared(a,suma,n,chunck)
  {  sumalocal=0;    tid=omp_get_thread_num();
     #pragma omp for private(i) schedule(static, chunck)
     for (i=0; i<n; i++)
     {   sumalocal += a[i];
        printf(" Thread %d suma de a[%d]=%d sumalocal=%d \n",tid,i,a[i],sumalocal);
     }
     #pragma omp atomic
        suma += sumalocal;
  }
  printf("Suma=%d\n", suma);
}
```

(a) (0.15) Indique qué orden usaría para compilar el código de `examen1.c` desde una ventana de comandos (*Shell* o intérprete de comandos) si el ejecutable se quiere llamar `examen1` (utilice en la compilación alguna de las opciones de optimización que ha utilizado en la práctica de optimización de código).

(b) (0.25) En el código hay tres puntos en los que se puede imprimir en pantalla: un `fprintf` y dos `printf`. Razone qué `fprintf y printf` de los que hay en el código se ejecutan y cuántas veces se ejecuta cada uno de ellos.

(c) (0.15) ¿Cuántos parámetros de entrada necesita el programa y para qué se utiliza cada uno de ellos en el código?

(d) (0.4) Escriba **todo** lo que imprime el programa (hay que poner **todo el texto que imprime un thread cada vez que imprime algo por pantalla**) si el usuario ejecuta (razone sus respuestas):

```
(1). examen1 4
(2). examen1 8  1
(3). examen1 8  2
```

(e) (0.25) Elimine del código todos los `default`, `private` y `shared` y realice en el código las modificaciones que sean estrictamente necesarias (sin añadir `default`, `private`, `firstprivate`, `lastprivate` y `shared`) para que imprima exactamente lo mismo que imprimía en la versión original en el último `printf` del código.

Cuestión 3.(0.3 puntos) Indique cuál de los siguientes códigos ofrece mejores prestaciones y explique los motivos:

```
for (i=0;i<100;i++)
    if ((i%2) = = 0)
          a[i]=x;
    else
         a[i]=y;
```

```
for (i=0;i<100;i+=2) {
      a[i]=x;
      a[i+1]=y; }
```

2 Solución Examen de Prácticas del 05/07/2013

Cuestión 1.(0.5 puntos) **(a)** (0.05) ¿Cuántos nodos de cómputo (servidores) tiene atcgrid y cuántos chips de procesamiento (encapsulados, procesador) tiene una placa madre de atcgrid? **(b)** (0.05) ¿Cuántos cores de procesamiento tiene un chip de procesamiento (encapsulado) de atcgrid? **(c)** (0.05) ¿Qué gestor de colas ha utilizado en prácticas para enviar trabajos a atcgrid? **(d)** (0.25) Suponga que debe ejecutar en atcgrid el fichero ejecutable `hello` que tiene en su home del PC del aula de prácticas (o en su PC portátil si está trabajando en el aula de prácticas con su propio portátil), ¿qué haría para ejecutarlo en atcgrid? ¿Cómo sabría que ya ha terminado la ejecución? ¿dónde podría consultar los resultados de la ejecución? **(e)** (0.1) Suponga que debe ejecutar en atcgrid el *script* `scripthello.sh` que tiene en su home del PC del aula de prácticas (o en su PC portátil si está trabajando en el aula de prácticas con su propio portátil), ¿qué haría para ejecutarlo?

Solución

(a) Tiene dos nodos de cómputo, atcgrid1 y atcgrid2. Cada placa madre tiene dos chips de procesamiento.

(b) Cada chip de procesamiento tiene 6 cores.

(c) En prácticas se ha utilizado el gestor de colas Torque.

(d) Una vez generado en el PC del aula de practicas el ejecutable `hello` para poderlo ejecutar en atcgrid haría lo siguiente:

a. Establecería desde mi PC local una conexión `sftp` y una conexión `ssh` con atcgrid.

b. Para trasladar el ejecutable a atcgrid, en el terminal con una conexión `sftp` a atcgrid que tengo en mi PC local, procedería de la siguiente forma:

   i. Me situaría en el PC local con `lcd` en el directorio donde tengo `hello`

   ii. Me situaría en mi home de atcgrid con `cd` en el directorio donde quiero ubicar `hello` (por ejemplo, en `ejhello`).

   iii. Ejecutaría `put hello` para trasladar `hello` a atcgrid desde el PC local

c. Una vez trasladado el fichero a atcgrid, en el terminal con conexión `ssh` a atcgrid que tengo en mi PC local, haría lo siguiente:

   i. Me situaría con `cd` en el directorio donde está `hello`

   ii. Ejecutaría (supongo que el ejecutable se encuentra en el directorio `ejhello`)

```
echo 'ejhello/hello' | qsub -q ac
```

   o ejecutaría `echo './ejhello/hello' | qsub -q ac`

d. Para comprobar si ha terminado la ejecución, en el terminal con conexión `ssh` a atcgrid que tengo en mi PC, utilizaría `qstat` para ver el estado de la ejecución (un C en la columna de estado indicaría que la ejecución se ha Completado) y el identificador asignado al proceso por Torque (el identificar se encuentra en la primera columna de la salida de `qstat`), y usaría `ls -lag` para comprobar si se han generado los ficheros de salida de Torque para el identificador que se le ha asignado (y que he podido descubrir con `qstat`).

e. Para consultar los resultados en mi PC local, primero trasladaría los ficheros desde atcgrid al PC local usando `get` en el terminal con conexión `sftp`: `get STDIN.oXXXX` y `get STDIN.eXXXX` (XXXX representa el identificador que Torque ha asignado al proceso). Después, usaría, por ejemplo, `more` o `cat` para listar el contenido del fichero de salida `STDIN.oXXXX` y el fichero de errores `STDIN.eXXXX`. Si alguno de los dos ficheros, el de salida o el de error, tiene tamaño 0 no consultaría su contenido, puesto que están vacíos. También podría editarlos con, por ejemplo, `gedit`.

(e) Para trasladar el fichero a atcgrid procedería de la misma forma que para trasladar el ejecutable del apartado (d), haría también finalmente la transferencia con `put`: `put scripthello.sh`. Para poder

ejecutar el *script* `scripthello.sh` que está en atcgrid haría lo siguiente (supongo que dentro del *script* no se especifica la cola de ejecución):

```
qsub -q ac scripthello.sh
```

◀◀◀◀◀◀◀◀◀◀◀◀◀◀◀◀◀◀◀

Cuestión 2. (0.5 puntos) Conteste a las siguientes cuestiones sobre el código `examen1.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>
int main(int argc, char **argv)
{  int i, n=20, tid, chunck;
   int a[n],suma=0,sumalocal;
   if(argc < 3)    {
     fprintf(stderr,"[ERROR]-Faltan datos de entrada\n");    exit(-1);
    }
   n = atoi(argv[1]); if (n>20) n=20; chunck = atoi(argv[2]);
   for (i=0; i<n; i++)    a[i] = i;

  #pragma omp parallel num_threads(4) default(none) private(sumalocal,tid) shared(a,suma,n,chunck)
  {  sumalocal=0;    tid=omp_get_thread_num();
     #pragma omp for private(i) schedule(static, chunck)
     for (i=0; i<n; i++)
     {   sumalocal += a[i];
        printf(" Thread %d suma de a[%d]=%d sumalocal=%d \n",tid,i,a[i],sumalocal);
     }
     #pragma omp atomic
        suma += sumalocal;
  }
  printf("Suma=%d\n", suma);
}
```

(a) (0.15) Indique qué orden usaría para compilar el código de `examen1.c` desde una ventana de comandos (*Shell* o intérprete de comandos) si el ejecutable se quiere llamar `examen1` (utilice en la compilación alguna de las opciones de optimización que ha utilizado en la práctica de optimización de código).

(b) (0.25) En el código hay tres puntos en los que se puede imprimir en pantalla: un `fprintf` y dos `printf`. Razone qué `fprintf y printf` de los que hay en el código se ejecutan y cuántas veces se ejecuta cada uno de ellos.

(c) (0.15) ¿Cuántos parámetros de entrada necesita el programa y para qué se utiliza cada uno de ellos en el código?

(d) (0.4) Escriba **todo** lo que imprime el programa (hay que poner **todo el texto que imprime un thread cada vez que imprime algo por pantalla**) si el usuario ejecuta (razone sus respuestas):

```
(4). examen1 4
(5). examen1 8  1
(6). examen1 8  2
```

(e) (0.25) Elimine del código todos los `default`, `private` y `shared` y realice en el código las modificaciones que sean estrictamente necesarias (sin añadir `default`, `private`, `firstprivate`, `lastprivate` y `shared`) para que imprima exactamente lo mismo que imprimía en la versión original en el último `printf` del código.

Solución

(a) Usaría: `gcc -O2 -fopenmp -o examen1 examen1.c`

(b) `fprintf` se podría ejecutar una vez o ninguna. El thread 0 (*master*) lo ejecuta si el número de parámetros de entrada al programa es menor que 3 (el usuario debe poner, cuando ejecuta el código, dos parámetros, `n` y `chunck`, detrás del nombre del ejecutable). No se ejecuta más de una vez porque está fuera de la región paralela. El `printf` que se encuentra dentro del bucle `for` se ejecuta `n` veces, tantas como iteraciones tiene el bucle. Aunque se encuentra dentro de la región paralela, las iteraciones del bucle se reparten entre los threads debido a que se usa la directiva `for`, por tanto, cada iteración sólo se va a ejecutar una vez, por este motivo, cada `printf` de una iteración sólo se va a ejecutar una vez (lo ejecuta un único thread). El último `printf`, que se encuentra fuera de la región paralela y que imprime justo antes de terminar el programa, sólo se ejecuta una vez ya que se encuentra fuera de la región paralela en lugar de dentro. Sólo lo ejecuta el thread 0 (*master*).

(c) Hay un total de 2 parámetros que requiere el programa como entrada. El primero se introduce en la variable `n` y el segundo en `chunck` (línea 10 del código). La variable `n` es el número de iteraciones del bucle `for` de la región paralela y, por tanto, el número de componentes del vector que se suma en paralelo. La variable `chunck` es el tamaño de la unidad que se asigna a threads. Con la directiva `for` estamos indicando a la herramienta que reparta las iteraciones del bucle entre los threads de forma que se ejecuten todas las iteraciones y sólo una vez cada una. Con la cláusula `schedule()` le estamos especificando además que debe realizar la asignación antes de la ejecución (porque se usa `static`) y el tamaño de las unidades que debe repartir. El tamaño es un número de iteraciones del bucle; es decir, si es 2, entonces las unidades serían de tamaño igual a dos iteraciones del bucle, y, si es 4, comprendería cuatro iteraciones cada una.

(d) (1). `examen1 4`

En este caso imprime "[ERROR]-Faltan datos de entrada" porque el número de parámetros de entrada al programa que se espera que introduzca el usuario es dos en lugar de uno. Debido a esto la condición del primer `if` del código se cumple y se ejecuta el código asociado al `if` que imprime en pantalla el mensaje comentado y termina la ejecución del código (línea 8 del código).

(2). `examen1 8 1`

En este caso se introducen los dos parámetros requeridos para la ejecución del código. El número de iteraciones del bucle `for`, o número de componentes del vector `a[]`, es 8 y el tamaño de `chunck` es 1. Los `printf` del bucle imprimen lo siguiente:

```
Thread 0 suma de a[0]=0 sumalocal=0
Thread 0 suma de a[4]=4 sumalocal=4
Thread 1 suma de a[1]=1 sumalocal=1
Thread 1 suma de a[5]=5 sumalocal=6
Thread 2 suma de a[2]=2 sumalocal=2
Thread 2 suma de a[6]=6 sumalocal=8
Thread 3 suma de a[3]=3 sumalocal=3
Thread 3 suma de a[7]=7 sumalocal=10
```

El orden en que se imprimen los mensajes de los `printf` de distintos threads puede variar entre una y otra ejecución. Los `printf` que imprime un threads aparecen en el orden en el que los ejecuta el thread (en orden creciente del índice `i`).

El `printf` fuera de la región paralela imprime: "Suma=28". Se imprime después que los mensajes que se generan dentro de la región paralela (los indicados más arriba). Se imprime después puesto que hay una barrera implícita al final de la región `parallel` en la que todos los threads se esperan entre ellos antes de dejar la región paralela. Una vez que todos han llegado a la barrera, el thread 0 continúa con la ejecución del código que hay detrás del bloque estructurado de la directiva `parallel` y el resto mueren.

(3). examen1 8 2

En este caso se introducen los dos parámetros necesarios. El número de iteraciones del bucle `for`, o número de componentes del vector `a[]`, es 8 y el tamaño de `chunck` es 2. Los `printf` del bucle imprimen lo siguiente:

```
Thread 0 suma de a[0]=0 sumalocal=0
Thread 0 suma de a[1]=1 sumalocal=1
Thread 1 suma de a[2]=2 sumalocal=2
Thread 1 suma de a[3]=3 sumalocal=5
Thread 2 suma de a[4]=4 sumalocal=4
Thread 2 suma de a[5]=5 sumalocal=9
Thread 3 suma de a[6]=6 sumalocal=6
Thread 3 suma de a[7]=7 sumalocal=13
```

El orden en que se imprimen los mensajes de los `printf` de distintos threads puede variar entre una y otra ejecución. Los `printf` que imprime un threads aparecen en el orden en el que los ejecuta el thread (en orden creciente del índice `i`).

Los `printf` fuera de la región paralela imprime: "Suma=28". Se imprime después que los mensajes que se generan dentro de la región paralela (los indicados más arriba) por el motivo ya comentado más arriba.

(e) Lo más sencillo es hacer lo siguiente: declarar las variables que necesariamente tienen que ser privadas dentro de la construcción `parallel` y el resto de variables fuera de la construcción `parallel` (el resto serían entonces todas compartidas). El código modificado se muestra en el siguiente recuadro:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char **argv)
{  int i, n=20, chunck;
  int a[n],suma=0;
  if(argc < 3)    {
    fprintf(stderr,"[ERROR]-Faltan datos de entrada\n");    exit(-1);
   }
  n = atoi(argv[1]); if (n>20) n=20; chunck = atoi(argv[2]);
  for (i=0; i<n; i++)    a[i] = i;
   #pragma omp parallel num_threads(4)
 {  int sumalocal=0;    int tid=omp_get_thread_num();
   #pragma omp for schedule(static, chunck)
   for (i=0; i<n; i++)
   {   sumalocal += a[i];
      printf(" Thread %d suma de a[%d]=%d sumalocal=%d \n",tid,i,a[i],sumalocal);
   }
   #pragma omp atomic
      suma += sumalocal;
 }
 printf("Suma=%d\n", suma);
}
```

Cuestión 3.(0.3 puntos) **(a)** Indique cuál de los siguientes códigos ofrece mejores prestaciones y explique los motivos:

```
for (i=0;i<100;i++)
    if ((i%2) = = 0)
          a[i]=x;
    else
         a[i]=y;
```

```
for (i=0;i<100;i+=2) {
      a[i]=x;
      a[i+1]=y; }
```

Solución

Ofrece mejores prestaciones el código de la derecha porque: (1) El número de iteraciones del bucle es la mitad ya que el índice se incrementa de dos en dos, lo que supone tener que ejecutar menos instrucciones de control del bucle (incrementar el índice, comprobar si se han realizado todas las iteraciones, saltar). (2) Se ha eliminado la instrucción para obtener el módulo (`i` % 2). (3) Se han eliminado las instrucciones necesarias para implementar `if-then-else` (algunas son de salto si el compilador no genera `cmov` para este código).