

Examen BP2 Resuelto.

Recuerda que me puedes seguir en wuolah :P

Pd: para cuando subo el examen, no se la nota ni los resultados, así que puede ser que alguna respuesta no sea correcta. Y otra cosa, no te fijas en las respuestas marcadas, fíjate en las respuestas.

¿Cuánto vale ret al final?

```
int i, n = 6, ret = 1;
#pragma omp parallel reduction(+:ret) private(i)
for (i=omp_get_thread_num(); i<n; i+=omp_get_num_threads())
    ret += i;
return ret;
```

SWAD: plataforma de apoyo a la docencia / UGR /

Usuaría Profesores

- ☐ a) 16
- ☐ b) 15
- ☐ c) Ninguna de las otras respuestas es correcta
- ☐ d) 1

Respuesta: a), ya que es como si hiciéramos un parallel for, y al final, se harán 6 iteraciones (de la 0 a la 5) y se sumarán a la variable ret, cuyo valor inicial es 1, todas las i ($1+0+1+2+3+4+5 = 16$).

Analiza el siguiente código e indica qué nos dirá el compilador referido a las siguientes líneas de código.

```
int i, n = 1;
#pragma omp parallel for default(none) private(i)
for (i = 0; i < 5; ++i)
    n += i;
```

Usuario Profesores

- ☐ a) La variable n no está especificada como privada
- ☐ b) Ninguna opción de las anteriores.
- ☐ c) La variable i no puede ser privada
- ☒ d) La variable n no está especificada

Respuesta: d) (puede ser que la quisieras especificar como compartida, y default(none) obliga a especificar el alcance de las variables usadas en la construcción, salvo índices de for y variables threadprivate).

¿Cuál de los siguientes fragmentos de código funciona de manera incorrecta?

Usuario Profesores

- ☐ a)

```
#pragma omp parallel for private(x)
for (long k = 0; k < 100; k++){
    x = array[k];
    array[k] = x+5;
}
```
- ☒ b) Todos los códigos funcionan de manera correcta
- ☐ c)

```
#pragma omp parallel for
for (long k = 0; k < 100; k++){
    int x;
    x = array[k];
    array[k] = x+5;
}
```
- ☐ d)

```
#pragma omp parallel for
for (long k = 0; k < 100; k++){
    x = array[k];
    array[k] = x+5;
}
```

Respuesta: d), el código de esta opción no es correcto ya que x sería una variable compartida al ser declarada previamente a la región paralela y, por tanto, su valor no lo podemos predecir.

¿Cuál de los siguientes fragmentos de código tardará menos en calcular la sumatoria de los primeros $N = 2^{25}$ números impares en una máquina con 4 procesadores?

Usuario Profesores

- ☒ a)

```
long sum = 0;
#pragma omp parallel sections
{
    #pragma omp section
    for (long i = 1; i < N; i += 4)
        #pragma omp atomic
        sum += i;
    #pragma omp section
    for (long i = 3; i < N; i += 4)
        #pragma omp atomic
        sum += i;
}
```
- ☒ b)

```
long sum = 0;
#pragma omp parallel
{
    long p = 0;
    #pragma omp for
    for (long i = 1; i < N; i += 2)
        p += i;
    #pragma omp atomic
    sum += p;
}
```
- ☐ c)

```
long sum = 0;
```

SWAD: plataforma de apoyo a la docencia / UGR /

```
#pragma omp parallel for
for (long i = 1; i < N; i += 2)
    #pragma omp atomic
    sum += i;
```

- ☒ d)

```
long sum = 0;
#pragma omp parallel sections
{
    #pragma omp section
    for (long i = 1; i < N; i += 2)
        #pragma omp atomic
        sum += i;
}
```

Respuesta: hemos pensado la c, al poder usarse todos los procesadores y ya que p no se pone a 0 tras haber entrado al bucle, causaría que se le sumase a sum una cantidad errónea (ej: p=1 en una iteración, y en la siguiente podría ser 4, al ser 3(i)+1(lo que tenemos acumulado en p)).

Si se quiere difundir el valor de una variable inicializada por una hebra en una directiva single a las variables del mismo nombre del resto de hebras. ¿Cuál sería la cláusula que se tendría que utilizar?

Usuario Profesores

- ☐ a) firstprivate
- ☐ b) lastprivate
- ☐ c) copyprivate
- ☐ d) Ninguna de las anteriores

Respuesta: c). *“Permite que una variable privada de un thread se copie a las variables privadas del mismo nombre del resto de threads(difusión)”.*

Indica cual será el valor de la variable n al final de la ejecución del siguiente código:

```
int n = 0;
#pragma omp parallel for reduction(*:n)
for (int i = n; i < size; ++i)
    n *= i;
```

Usuario Profesores

- ☐ a) Dependerá del valor de la variable size.
- ☐ b) Ninguna respuesta es correcta.
- ☐ c) El valor de n será igual a size -1.
- ☐ d) 0

Respuesta: a), ya que reduction haría con los resultados parciales de todas las hebras, lo mismo que hizo para obtener el resultado parcial. Así, n valdrá más o menos en función del número de iteraciones.

Analizando el siguiente código, ¿se puede asegurar que las componentes del vector c, estarán todas calculadas al finalizar la región paralela?

```
int main(int argc, char **argv) {
    int sum = 0, i, N = 10, a[10], b[10], c[10];
    #pragma omp parallel
    {
        #pragma omp sections
        {
            #pragma omp section
            for (i = 0; i < N/2; i++){
                c[i]=a[i]+b[i];
            }

            #pragma omp section
            for (i = N/2; i < N; i++){
                c[i]=a[i]+b[i];
            }
        }
    }
}
```

☐ a) Sí, siempre que haya al menos dos hebras.

☐ b) No, porque si se ejecuta con una sola hebra, no se ejecutarán las dos secciones.

☐ c) Sí, entre las dos secciones, se calcula todo.

☐ d) No, porque es posible que no se ejecuten todas las iteraciones del bucle.

Respuesta: c), ya que las regiones sections, siempre se ejecutarán, dando igual el número de hebras que haya. (ej: si tenemos 1 hebra, esta ejecutará las dos sections, si tenemos 2 o más, 1 hebra ejecutará cada sections).

¿Existe algún problema que impida compilar el siguiente código?

```
int n = 1;
#pragma omp parallel for default(none)
for (int i = 0; i < 5; ++i)
    n += i;
```

Usuario Profesores

- ☒ a) Existe una condición de carrera
- ☒ b) El ámbito de la variable n no está definido.
- ☒ c) Ninguna otra respuesta es correcta
- ☒ d) El ámbito de la variable i no está definido

Respuesta: b), al tener que especificar el ámbito de n. (tipo de pregunta explicada previamente).

Supongamos una máquina en la que el número de hebras de las que se puede disponer para ejecutar zonas paralelas de código es ilimitado. En esas condiciones, ¿qué valdrá la variable n al terminar de ejecutar el siguiente código?

```
int n = 1;
#pragma omp parallel for firstprivate(n) lastprivate(n)
for (int i = 0; i < 5; ++i)
    n += i;
```

Usuario Profesores

- ☐ a) Dependerá del número de hebras que lo ejecuten en cada momento.
- ☐ b) n al salir del bucle está indefinida porque es privada.
- ☐ c) 1
- ☒ d) 5

Respuesta: a), ya que puede ocurrir como en el ejercicio 4 de la relación. (si tenemos hebras infinitas, correspondería una iteración a cada hebra, y la respuesta sería la d), pero como hay una opción que dice claramente que depende del número de hebras, esta es más correcta, ya que puede ser que una hebra haga varias iteraciones y por tanto, el valor de n sería distinto al que tendría si la hebra que hiciese la última iteración, tuviese solo esa)

Observando el siguiente código. ¿Cuál es el error o los errores para que el programa compile correctamente y obtenga el resultado deseado?

```
int n = 1, constant;
#pragma omp parallel for reduction(*n) private(constant)
for (int i = 0; i < omp_get_max_threads(); ++i)
    n *= (i + constant);
return n;
```

- ☐ a) Todas las otras respuestas son correctas
- ☐ b) La variable `constant` debe ser inicializada
- ☐ c) Hay que cambiar la cláusula `private` por `firstprivate`
- ☐ d) La cláusula `reduction` tiene una sintaxis errónea.

Respuesta: a), ya que la sintaxis de `reduction` debe incluir `:`, luego, `constant` tiene basura (no hay `firstprivate` + asignación previa ni hay una sentencia que especifique para cada hebra su valor) y sabiendo que hay dos correcta, la respuesta debe ser la a). (cabe decir que la c) puede dar lugar a interpretación ya que `constant` no tiene valor asignado previo al bloque paralelo).

Indica si el valor que tomará la variable `v` al salir de la zona paralela será siempre el mismo para cualquier entorno de ejecución.

```
int main() {
    int i, n = 7, v = 1;
    int a[n];
    for (i=0; i<n; i++)    a[i] = i;
    #pragma omp parallel for firstprivate(v) lastprivate(v)
    for (i=0; i<n; i++){
        v += a[i];
        printf("\nthread %d v=%d / ", omp_get_thread_num(), v);
    }
    printf("\nFuera de la construcción parallel for v=%d\n", v);
    return 0;
}
```

SWAD: plataforma de apoyo a la docencia / UGR /

Usuario Profesores

- ☐ a) Sí, será siempre el mismo, 7
- ☐ b) No, será diferente dependiendo del número de hebras que lo ejecuten.
- ☐ c) Sí, será siempre el mismo, aunque cambie el número de hebras
- ☐ d) No, porque al salir de la zona paralela la variable volverá a ser 1.

Respuesta: b), por lo especificado en una pregunta previa sobre el número de hebras.


```
¿Cuánto vale n al final?  
int n = 1;  
#pragma omp parallel  
{  
    int p = 0;  
    #pragma omp single copyprivate(p)  
        p = 2;  
    #pragma omp for reduction(+:n)  
    for (int i = 0; i < 5; ++i)  
        n += p;  
}  
return n;  
Usuario Profesores
```

- ☒ a) 11
☐ b) 10
☐ c) 5
☐ d) 1

SWAD: plataforma de apoyo a la docencia / UGR /

- ☐ c) 5
☐ d) 1

Respuesta: a), ya que tiene valor 1 inicialmente para todas las hebras y luego se hacen 5 iteraciones en el for, en las que da igual el número de hebras que haya, porque hay una cláusula reduction que suma el valor de todas las sumas parciales que se hayan podido crear.

Sobre el código que aparece a continuación, ¿qué afirmación es correcta?

```
#pragma omp parallel private(sumalocal)
{
    sumalocal = 0;
    #pragma omp for
    for (i=0; i<n; i++)
        sumalocal += a[i];
    #pragma omp barrier
    #pragma omp critical
        suma = suma + sumalocal;
    #pragma omp barrier
    #pragma omp single
        printf("La suma es %d\n", suma);
}
```

SWAD: plataforma de apoyo a la docencia / UGR /

Usuario Profesores

- ☐ a) Todas las demás respuestas son incorrectas
- ☐ b) Uno de los `#pragma omp barrier` es innecesario
- ☐ c) El valor de `suma` que se imprime sería correcto si cambiamos `private(sumalocal)` por `private(sumalocal, suma)`
- ☐ d) El valor de `suma` que se imprime no es siempre correcto

Respuesta: b), ya que el primer `barrier` es innecesario ya que `#pragma omp for` tiene barrera implícita al final. (d) es falsa, ya que tal y como está actualmente, si da el resultado correcto y respecto a c), si ponemos `suma` como privada, cada hebra tendrá su propia suma y no sería la suma de todas las hebras).

Indica qué valor tendrá la variable `ret` después de ejecutar la siguiente reducción cuando se ejecuta con 4 hebras:

```
int i, n = 6, ret = 1;
#pragma omp parallel reduction(+:ret)
for (i=omp_get_thread_num(); i<omp_get_max_threads(); i+=omp_get_num_threads())
    ret += i;
```

Usaria Profesores

- ☐ a) 7
- ☐ b) El valor de `ret` será indeterminado porque existe una condición de carrera
- ☐ c) 5
- ☐ d) 3

Respuesta: sería la b), ya que hay una condición de carrera con la variable `i`, ya que todas las modificarán. (f por mí 😞).

¿Cuál será el valor de n tras ejecutar el siguiente código?

```
int i, n=2;
#pragma omp parallel shared(n) private(i)
  for(i=0; i < 4; i++){
    #pragma omp single
    {
      n += i;
    }
  }
```

Usaria Profesores

- ☐ a) 16
- ☐ b) Indeterminado
- ☐ c) 2
- ☒ d) 8

Respuesta: d), ya que es como un parallel for, donde n se comparte y tiene valor inicial 2 (que se comparte por el shared) y al haber un single, impedimos que haya condiciones de carrera con la variable n y, por tanto, $n=2+0+1+2+3=8$.

¿Qué valdrá la variable n después de la ejecución del siguiente código?

```
int n = -1;
#pragma omp parallel for lastprivate(n)
for (int i = 0; i < 4; ++i)
  #pragma omp atomic
  n += i;
```


Usaria Profesores


- ☐ a) 5
- ☐ b) 2
- ☒ c) Dependerá del número de hebras
- ☐ d) 3


Respuesta: c), (especificado que hace lastprivate antes).

Asumiendo que $v2$ es de dimensión N y que todos sus elementos están inicializados a cero, ¿cuál de los siguientes códigos calcula de forma correcta el producto de la matriz m (dimensión $N \times N$) por el vector $v1$ (dimensión N) paralelizando el bucle que recorre las columnas?

Usuaría Profesores


-  a)

```
#pragma omp parallel private(i,j)
for(i=0;i<N;i++){
    for(j=0;j<N;j++){
        v2[i] += m[i][j]*v1[j];
    }
}
```
-  b)

```
#pragma omp parallel private(j)
for(i=0;i<N;i++){
    #pragma omp for reduction(+:v2[i])
    for(j=0;j<N;j++){
        v2[i] += m[i][j]*v1[j];
    }
}
```
-  c)

```
#pragma omp parallel private(i)
for(i=0;i<N;i++){
    #pragma omp for reduction(+:v2[i])
    for(j=0;j<N;j++){
        v2[i] += m[i][j]*v1[j];
    }
}
```

SWAD: plataforma de apoyo a la docencia / UGR /

-  d)

```
#pragma omp parallel for private(j)
for(i=0;i<N;i++){
    for(j=0;j<N;j++){
        v2[i] += m[i][j]*v1[j];
    }
}
```

Respuesta: c), ya que hay que poner la variable i como privada y luego, sumar (reduction) para ver el valor de ese elemento de la matriz. (otra f por mí, pero media f porque no puse ninguna xd).

¿Cuáles de las siguientes cláusulas crean instancias privadas de una variable con un valor indefinido?

Usuario Profesores

- ☐ a) `copyprivate` y `firstprivate`
- ☐ b) `private` y `firstprivate`
- ☐ c) `firstprivate` y `lastprivate`
- ☐ d) `private` y `lastprivate`

Respuesta: d), ya que `firstprivate`, asigna el valor inicial de la variable a todas las hebras.