

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Antonio Gámiz Delgado

Grupo de prácticas: B

Fecha de entrega: 13/04/2018

Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

1. ¿Qué ocurre si en el ejemplo del seminario `shared-clause.c` se añade a la directiva `parallel` la cláusula `default(none)`? (añada una captura de pantalla que muestre lo que ocurre) **(b)** Resuelva el problema generado sin eliminar `default(none)`. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA:

Al añadir `default(none)` a la cláusula `parallel` e intentar compilar, vemos que nos da un error. Nos dice que la variable "n" no está especificada dentro del bloque paralelo. Esto se debe a que `default(none)` indica que el programador debe ser el que indique al bloque paralelo mediante código a qué variables pueden tener acceso los threads. Esto se puede arreglar haciendo lo dicho anteriormente, añadiendo la cláusula `shared(n)`.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

```
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#endif

int main()
{
    int i, n=7;
    int a[n];

    for(i=0; i<n; i++) a[i]=i+1;

    #pragma omp parallel for shared(a) default(none) shared(n)
    for(i=0; i<n; i++) a[i]+=i;

    printf("Después de parallel for:\n");
    for(i=0; i<n; i++) printf("a[%d]=%d\n", i, a[i]);
}
```

CAPTURAS DE PANTALLA:

```
antoniogamizdelgado 2018-04-11 miércoles @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/shared-clauseModificado.c -o ./bin/shared-clauseModificado
./source/shared-clauseModificado.c: In function 'main':
./source/shared-clauseModificado.c:17:13: error: 'n' not specified in enclosing parallel
    #pragma omp parallel for shared(a) default(none)
                    ^
./source/shared-clauseModificado.c:17:13: error: enclosing parallel
```

```
antoniogamizdelgado 2018-04-11 miércoles @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/shared-clauseModificado
Después de parallel for:
a[0]=1
a[1]=3
a[2]=5
a[3]=7
a[4]=9
a[5]=11
a[6]=13
antoniogamizdelgado 2018-04-11 miércoles @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
```

2. ¿Qué ocurre si en `private-clause.c` se inicializa la variable `suma` fuera de la construcción `parallel` en lugar de dentro? (inicialice `suma` a un valor distinto de 0 dentro y fuera de `parallel`) Razone su respuesta. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA:

Vemos que al inicializar la variable `suma` a dos valores distintos entre sí (-2 y -1), uno fuera de la región paralela (-2) y el otro dentro (-1), el primero es ignorado y sobrescrito por el segundo, produciendo un resultado erróneo de la suma. Esto se debe a la cláusula `private`, que hace como si redecláramos la variable `suma` dentro de la región paralela para cada thread, es decir, cada thread tiene su propia variable `suma` independiente (si dentro de la región no inicializamos `suma` entonces tendrá un valor basura).

```
antoniogamizdelgado 2018-04-11 miércoles @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/private-clauseModificado
thread 1 suma a[2]
thread 1 suma a[3]
thread 3 suma a[6]
thread 2 suma a[4]
thread 2 suma a[5]
thread 0 suma a[0]
thread 0 suma a[1]

* thread 2 suma=8
* thread 1 suma=4
* thread 0 suma=0
* thread 3 suma=5
```

CAPTURA CÓDIGO FUENTE: private-clauseModificado.c

```

#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main()
{
    int i, n=7;
    int a[n], suma;

    for(i=0; i<n; i++) a[i]=i;

    suma=-2;
    #pragma omp parallel private(suma)
    {
        suma=-1;
        #pragma omp for
        for(i=0; i<n; i++)
        {
            suma+=a[i];
            printf("thread %d suma a[%d]\n", omp_get_thread_num(), i);
        }
        printf("\n* thread %d suma=%d", omp_get_thread_num(), suma);
    }
    printf("\n");
}

```

3. ¿Qué ocurre si en private-clause.c se elimina la cláusula `private(suma)`? ¿A qué cree que es debido?

RESPUESTA:

Ocurre que, como vemos. La suma siempre sale igual a 15.

Esto es debido a que, como he dicho en el ejercicio anterior, `private` hace que cada thread tenga su propia variable `suma`, pero en este caso, todos los thread están actuando sobre la misma variable `suma`, por lo que además de realizar la suma mal, todos muestran lo mismo. El fallo en la suma se debe a un fallo de dependencia de datos y el de que muestren lo mismo al haber solo una variable `suma` todos muestran la misma (la del proceso).

```

antoniogamizdelgado 2018-04-11 miércoles @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/private-clauseModificado2
thread 0 suma a[0]
thread 0 suma a[1]
thread 3 suma a[6]
thread 2 suma a[4]
thread 2 suma a[5]
thread 1 suma a[2]
thread 1 suma a[3]

* thread 0 suma=15
* thread 2 suma=15
* thread 1 suma=15
* thread 3 suma=15
antoniogamizdelgado 2018-04-11 miércoles @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)

```

CAPTURA CÓDIGO FUENTE: private-clauseModificado3.c

```

#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main()
{
    int i, n=7;
    int a[n], suma;

    for(i=0; i<n; i++) a[i]=i;

    #pragma omp parallel
    {
        suma=0;
        #pragma omp for
        for(i=0; i<n; i++)
        {
            suma+=a[i];
            printf("thread %d suma a[%d]\n", omp_get_thread_num(), i);
        }
        printf("\n* thread %d suma=%d", omp_get_thread_num(), suma);
    }
    printf("\n");
}

```

4. En la ejecución de firstlastprivate.c de la pag. 21 del seminario se imprime un 6 fuera de la región parallel. ¿El código imprime siempre 6 fuera de la región parallel? Razone su respuesta.

RESPUESTA:

Sí, ya que aunque esté puesto firstprivate, el último que se ejecuta es lastprivate, por lo que firstprivate es como si no estuviera.

(En la captura en lugar de 6 aparece 11 porque la carga de trabajo se ha repartido diferente en mis threads que en los de la transparencia).

CAPTURAS DE PANTALLA:

```

$ gcc -fopenmp -O2 ./source/firstlastprivate-clause.c -o ./bin/firstlastprivate-clause
antonio.gamizdelgado 2018-04-11 miércoles @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
$ ./bin/firstlastprivate-clause
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 2 suma a[5] suma=5
thread 2 suma a[6] suma=11
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7

Fuera de la construcción 'parallel for' suma=11

```

5. ¿Qué se observa en los resultados de ejecución de `copyprivate-clause.c` cuando se elimina la cláusula `copyprivate(a)` en la directiva `single`? ¿A qué cree que es debido?

RESPUESTA:

En la captura vemos que solo algunos valores del vector han sido correctamente inicializados, en concreto siempre valores consecutivos. Esto se debe a que al eliminar la directiva, el valor de la variable “a” no ha sido copiada a los demás threads, por lo que el único thread que tiene el valor correcto de “a” es el que ha ejecutado la sentencia `single`, en este caso el thread 1, que es el que ha ejecutado las iteraciones 3, 4 y 5.

```
antoniogamizdelgado 2018-04-11 miércoles @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/copyprivate-clauseModificado.c -o ./bin/copyprivate-clauseModificado
antoniogamizdelgado 2018-04-11 miércoles @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/copyprivate-clauseModificado

Introduce el valor de inicializacion a:1

Single ejecutada por el thread 1
Despues de la region parallel:
b[0]=0 b[1]=0 b[2]=0 b[3]=1 b[4]=1 b[5]=1 b[6]=0 b[7]=0 b[8]=0
antoniogamizdelgado 2018-04-11 miércoles @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶
```

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```
#include <stdio.h>
#include <omp.h>

int main(){
    int n=9, i, b[n];

    for(i=0; i<n; i++) b[i]=-1;

    #pragma omp parallel
    {
        int a;
        #pragma omp single
        {
            printf("\nIntroduce el valor de inicializacion a:");
            scanf("%d", &a);
            printf("\nSingle ejecutada por el thread %d \n", omp_get_thread_num());
        }

        #pragma omp for
        for(i=0; i<n; i++) b[i]=a;
    }

    printf("Despues de la region parallel:\n");
    for(i=0; i<n; i++) printf("b[%d]=%d\t", i, b[i]);
    printf("\n");
}
```

6. En el ejemplo `reduction-clause.c` sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado

RESPUESTA:

En el caso de ejecutarlo con $n=6$, se obtiene 25, es decir, la suma correcta (15) más el valor al que lo hemos inicializado (10). Esto se debe a que al aplicar la directiva `reduction`, la variable `suma` pasa a ser local en cada uno de los threads, es decir, cada thread tiene su variable `suma_i`, que al finalizar se suman todas en la variable `suma` definida arriba, que es global. Por eso, siempre saldrá el resultado correcto de la suma más el valor al que hayamos inicializado la variable `suma`.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

CAPTURAS DE PANTALLA:

7. En el ejemplo `reduction-clause.c`, elimine `reduction()` de `#pragma omp parallel for reduction(+:suma)` y haga las modificaciones necesarias para que se siga realizando la suma de los componentes del vector `a` en paralelo sin usar directivas de trabajo compartido.

RESPUESTA:

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

CAPTURAS DE PANTALLA:

Resto de ejercicios

8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, `M`, por un vector, `v1` (implemente una versión para variables globales y otra para variables dinámicas, use una de estas versiones en los siguientes ejercicios):

$$v2 = M \bullet v1; \quad v2(i) = \sum_{k=0}^{N-1} M(i, k) \bullet v(k), \quad i = 0, \dots, N-1$$

NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada al programa; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, `v3`, para tamaños pequeños de los vectores (por ejemplo, $N = 8$ y $N=11$); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que calcula el producto matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

CAPTURA CÓDIGO FUENTE: `pmv-secuencial.c`

CAPTURAS DE PANTALLA:

9. Implementar en paralelo el producto matriz por vector con OpenMP a partir del código escrito en el ejercicio anterior usando la directiva `for`. Debe implementar dos versiones del código (consulte la lección 5/Tema 2):
- una primera que paralelice el bucle que recorre las filas de la matriz y
 - una segunda que paralelice el bucle que recorre las columnas.

Use las directivas que estime oportunas y las cláusulas que sean necesarias **excepto la cláusula `reduction`**. Se debe paralelizar también la inicialización de las matrices. Respecto a este ejercicio:

- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, `v3`, para tamaños pequeños de los vectores (por ejemplo, $N = 8$ y $N=11$); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código que calcula el producto matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

CAPTURA CÓDIGO FUENTE : `pmv-OpenMP-a.c`

CAPTURA CÓDIGO FUENTE: `pmv-OpenMP-b.c`

RESPUESTA:

CAPTURAS DE PANTALLA:

10. A partir de la segunda versión de código paralelo desarrollado en el ejercicio anterior, implementar una versión paralela del producto matriz por vector con OpenMP que use para comunicación/sincronización la cláusula `reduction`. Respecto a este ejercicio:

- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

CAPTURA CÓDIGO FUENTE: `pmv-OpenmMP-reduction.c`

RESPUESTA:

CAPTURAS DE PANTALLA:

11. Ayudándose de una hoja de cálculo (recuerde que en las aulas está instalado OpenOffice) realice una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en `atcgrid` y en su PC del mejor código paralelo de los tres implementados en los ejercicios anteriores para dos tamaños (N) distintos (consulte la Lección 6/Tema 2). Usar `-O2` al compilar. Justificar por qué el código escogido es el mejor. NOTA: Nunca ejecute en `atcgrid` código que imprima todos los componentes del resultado.

CAPTURAS DE PANTALLA (que justifique el código elegido):

TABLA Y GRÁFICA (por *ejemplo* para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: un N entre 30000 y 100000, y otro entre 5000 y 30000):

COMENTARIOS SOBRE LOS RESULTADOS: