Grai2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Elena Merelo Molina
Grupo de prácticas: 2
Fecha de entrega:
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:
Genera un error, ya que al poner la cláusula default(none) las variables de fuera de la región paralela dejan de ser compartidas , que es su comportamiento por defecto, se establece que comparten a pero no saben qué hacer con n:

```
2018-04-11 16:38:27  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP2/src
→ gcc -lrt -O2 -fopenmp shared_clause.c -o ../bin/shared_clause
shared_clause.c: In function 'main':
shared_clause.c:14:11: error: 'n' not specified in enclosing parallel
   #pragma omp parallel for default(none) shared(a)
           ^
shared_clause.c:14:11: error: enclosing parallel
```

Así pues, para que deje de dar error simplemente hemos de especificar cómo queremos que sea n para los threads:

```
#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 default(none) shared(a) 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]);
}
```

Hemos considerado n como shared ya que si no la suma no se realiza correctamente, de

hecho ni entra en la región paralela, como se aprecia en la siguiente captura, que muestra el resultado de ejecutar considerando n privada y luego el resultado al recompilar con n compartida:

```
○ → ../bin/shared_clause
Después de parallel for:
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
a[5] = 6
a[6] = 7

 2018-04-11 16:44:15  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → gcc -lrt -O2 -fopenmp shared_clause.c -o ../bin/shared_clause

 2018-04-11 16:44:25  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → ../bin/shared_clause
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
```

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: Si inicializamos suma a un valor distinto de 0 dentro de parallel:

```c
//Elena Merelo Molina

#include <stdio.h>

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

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

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

  #pragma omp parallel private(suma)
  {
    suma=10;
    #pragma omp for
    for (i=0; i<n; i++){
      suma += a[i];
      printf("thread %d suma a[%d] / ", omp_get_thread_num(), i);
    }
    printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
  }
}
```

Al ejecutar resulta lo mismo, sumado 10 unidades, que antes de haber cambiado 0 por 10:

```
 2018-04-11 16:59:46  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → ../bin/private_clause
thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 0 suma a[3]
 / thread 1 suma a[4] / thread 1 suma a[5] / thread 1 suma a[6] /
* thread 0 suma= 6
* thread 1 suma= 15
 2018-04-11 17:00:12  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → gcc -lrt -O2 -fopenmp private_clause.c -o ../bin/private_clause

 2018-04-11 17:03:48  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → ../bin/private_clause
thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 0 suma a[3]
 / thread 1 suma a[4] / thread 1 suma a[5] / thread 1 suma a[6] /
* thread 0 suma= 16
* thread 1 suma= 25
```

Si sacamos suma= 10 de la región parallel:

```c
//Elena Merelo Molina

#include <stdio.h>

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

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

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

  suma=10;

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

Al ejecutar, sea cual sea el valor de suma siempre muestra:

```
 2018-04-11 17:03:50  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → gcc -lrt -O2 -fopenmp private_clause.c -o ../bin/private_clause

 2018-04-11 17:06:17  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → ../bin/private_clause
thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 0 suma a[3]
 / thread 1 suma a[4] / thread 1 suma a[5] / thread 1 suma a[6] /
* thread 0 suma= 8
* thread 1 suma= 4196495
```

Está sumando basura a suma, al haberla puesto privada cada hebra tiene un valor de suma propia, que no estar inicializado dentro de la región parallel es cualquiera.

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

RESPUESTA: Si eliminamos private(suma), el thread 0 y el 1 devuelven un resultado de la suma que es igual, ponga donde ponga suma(dentro o fuera de la región paralela), la inicialice a lo que la inicialice, ya que comparten esa variable por defecto:

```c
//Elena Merelo Molina

#include <stdio.h>

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

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

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


  suma=0;

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

Tras compilar y generar el ejecutable:

```
 2018-04-11 17:19:46  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → ../bin/private_clause_3
thread 0 suma a[0], suma= 202 / thread 0 suma a[1], suma= 0 / thread 0 suma a[2],
 suma= 0 / thread 0 suma a[3], suma= 0 / thread 1 suma a[4], suma= 202 / thread 1
 suma a[5], suma= 0 / thread 1 suma a[6], suma= 0 /
* thread 0 suma= 21
* thread 1 suma= 21
```

4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 7 fuera de la región `parallel`. ¿El código imprime siempre 6 fuera de la región `parallel`? Razone su respuesta.

RESPUESTA: Imprime siempre 7, que es la última componente del vector, ya que con lastprivate, en el caso de que estemos en un parallel for, imprime el resultado de la última iteración, y ésta es siempre a[6] = 7.

CAPTURAS DE PANTALLA:

```
 2018-04-11 17:21:21  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → gcc -lrt -O2 -fopenmp lastprivate_clause.c -o ../bin/lastprivate_clause

 2018-04-11 17:37:47  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → ../bin/lastprivate_clause
thread 0 v=1 / thread 0 v=2 / thread 0 v=3 / thread 0 v=4 / thread 1 v=5 / thread
 1 v=6 / thread 1 v=7 /
Fuera de la construcción'parallel for' v=7
```

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?

Si compilamos y ejecutamos sin quitar el copyprivate(a) obtenemos:

```
 2018-04-11 17:37:53  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → gcc -lrt -O2 -fopenmp copyprivate_clause.c -o ../bin/copyprivate_clause

 2018-04-11 17:46:34  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → ../bin/copyprivate_clause

Introduce valor de inicialización a: 3

Single ejecutada por el thread 0
Depués de la región parallel:
b[0] = 3        b[1] = 3        b[2] = 3        b[3] = 3        b[4] = 3        b
[5] = 3 b[6] = 3        b[7] = 3        b[8] = 3
```

Quitándolo:

```
 2018-04-11 17:51:46  elena in ~/Escritorio/University stuff
/2º/2º Cuatrimestre/AC/Prácticas/BP2/src
○ → ../bin/copyprivate_clause

Introduce valor de inicialización a: 3

Single ejecutada por el thread 0
Depués de la región parallel:
b[0] = 3        b[1] = 3        b[2] = 3        b[3] = 3
        b[4] = 3        b[5] = 0 b[6] = 0        b[7] = 0
        b[8] = 0
```

Observamos como algunas componentes se han inicializado y otras no, ya que el valor de a no ha llegado a todas las componentes. Copyprivate toma el valor o valores que pasados como parámetro y hace un “broadcast” de ellos de un thread a otro, después de la ejecución de lo que haya en el bloque single y antes de llegar a la barrera implícita. Por eso, al no ponerlo el thread que sí ha inicializado a no lo transmite al resto, quedándose con el valor 0.

Código fuente tras la modificación:

```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 copyprivate(a)
    {
      printf("\nIntroduce valor de inicialización 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("Depués de la región 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: Imprime el resultado que imprimía cuando suma= 0 sumándole 10.
Resultado de compilar y ejecutar cuando suma era 0:

```
 2018-04-11 18:10:51  elena in ~/Escritorio/University stuff/2º
/2º Cuatrimestre/AC/Prácticas/BP2/src
◯ → ../bin/reduction_clause 5
Tras 'parallel' suma=10

 2018-04-11 18:10:53  elena in ~/Escritorio/University stuff/2º
/2º Cuatrimestre/AC/Prácticas/BP2/src
◯ → ../bin/reduction_clause 3
Tras 'parallel' suma=3

 2018-04-11 18:10:58  elena in ~/Escritorio/University stuff/2º
/2º Cuatrimestre/AC/Prácticas/BP2/src
◯ → ../bin/reduction_clause 6
Tras 'parallel' suma=15
```

Cambiando el valor de suma por 10:

```
 2018-04-11 18:11:01  elena in ~/Escritorio/University stuff/2º/
2º Cuatrimestre/AC/Prácticas/BP2/src
○ → gcc -lrt -O2 -fopenmp reduction_clause.c -o ../bin/reduction_c
lause

 2018-04-11 18:11:22  elena in ~/Escritorio/University stuff/2º/
2º Cuatrimestre/AC/Prácticas/BP2/src
○ → ../bin/reduction_clause 6
Tras 'parallel' suma=25

 2018-04-11 18:11:25  elena in ~/Escritorio/University stuff/2º/
2º Cuatrimestre/AC/Prácticas/BP2/src
○ → ../bin/reduction_clause 3
Tras 'parallel' suma=13

 2018-04-11 18:12:23  elena in ~/Escritorio/University stuff/2º/
2º Cuatrimestre/AC/Prácticas/BP2/src
○ → ../bin/reduction_clause 5
Tras 'parallel' suma=20
```

Esto es ya que con la cláusula reduction OpenMP crea un conjunto de hebras entre las que divide las iteraciones del for. Cada hebra tiene su propia copia local de la variable de dentro de la cláusula, en este caso suma, y es ésta la que modifica. Cuando todas las hebras se juntan (reducción es un caso de paralelismo todos a uno), todas las copias locales se combinan en la variable global compartida.

Consecuentemente, en este programa cada hebra tiene la suma de los a[i] que le toquen, y cuando todas han realizado sus sumas locales se suma a su vez a la suma que tenemos declarada fuera de la región paralela.

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: Usamos la directiva parallel, poniendo suma_local como privada para que cada thread sume, dentro del for paralelo, su valor de suma, y una vez hecho esto vayan sumando una a uno su valor local a la variable global, con atomic, y se obtiene lo mismo que usando la cláusula reduction e inicializando la suma a 10:

```
 2018-04-13 12:49:48  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → gcc -fopenmp -lrt reduction_clause_modif.c -o ../bin/reduction_clause_modif

 2018-04-13 12:49:54  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → ../bin/reduction_clause_modif 6
Tras 'parallel' suma=25

 2018-04-13 12:49:56  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → ../bin/reduction_clause_modif 3
Tras 'parallel' suma=13

 2018-04-13 12:49:58  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP2/src
○ → ../bin/reduction_clause_modif 5
Tras 'parallel' suma=20
```

Código fuente modificado:

```c
//Elena Merelo Molina
#include <stdio.h>
#include <stdlib.h>

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

int main(int argc, char **argv) {
  int i, n=20, a[n],suma= 10, suma_local;

  if(argc < 2) {
    fprintf(stderr, "Faltan iteraciones\n");
    exit(-1);
  }

  n = atoi(argv[1]);

  if (n>20) {
    n=20;
    printf("n=%d",n);
  }

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

  #pragma omp parallel private(suma_local)
  {
    suma_local= 0;

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

    #pragma omp atomic
      suma += suma_local;
  }
  printf("Tras 'parallel' suma=%d\n",suma);
}
```

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):

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

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):
    a. una primera que paralelice el bucle que recorre las filas de la matriz y
    b. 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: