2º 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): Ivan Ortega Alba
Grupo de prácticas: C2
Fecha de entrega: 29/04/2015
Fecha evaluación en clase: 30/04/2015

---

1. **(a)** ¿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:

   Que todas las variables que se usan dentro del `parallels` y han sido declaradas anteriomente dejan de ser `shared` por defecto.

   Por tanto ahora tienen que ser declaradas explícitamente con clausulas, `private` o `shared` por ejemplo.

   CÓDIGO FUENTE: `shared-clauseModificado.c`

```
#include <stdio.h>

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

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)
   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:

```
cvi066023:1 ivanortegaalba$ gcc -fopenmp -O2 shared-clause.c -o SharedDefault
shared-clause.c: In function 'main':
shared-clause.c:19:12: error: 'n' not specified in enclosing parallel
    #pragma omp parallel for shared(a) default(none)
            ^
shared-clause.c:19:12: error: enclosing parallel
```

2. ¿Qué ocurre si en `private-clause.c` se inicializa la variable `suma` fuera de la construcción `parallel` en lugar de dentro? Razone su respuesta. Añada el código con la modificación al cuaderno de prácticas.

   RESPUESTA:

   El valor de la variable `private` al entrar al `parallels` es indefinido, por tanto, usaremos

un valor erróneo de leer esa variable dentro de este bloque.

CÓDIGO FUENTE: `private-clauseModificado.c`

```
#include <stdio.h>

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

main()
{
   int i, n = 7;
   int a[n], suma=50;

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

   suma=0;
   #pragma omp parallel private(suma)
   {
             #pragma omp for
             for (i=0; i<n; i++)
             {
                suma = 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);
    }

     printf("\n");
     printf("suma=%d \n",suma);
}
```

CAPTURAS DE PANTALLA:

```
cvi066023:2 ivanortegaalba$ ./PrivateBeforeParallels
thread 1 suma a[2] / thread 0 suma a[0] / thread 2 suma a[4] / thread 3 suma a[6
] / thread 1 suma a[3] / thread 0 suma a[1] / thread 2 suma a[5] /
* thread 3 suma= 163655094
* thread 0 suma= 5
* thread 1 suma= 163655093
* thread 2 suma= 163655097
suma=0
cvi066023:2 ivanortegaalba$
```

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

   RESPUESTA:

   La variable suma es compartida, por lo que todas las hebras escriben sobre esta y se pisan una a otra, por lo que sale un valor erróneo de la suma.

   CÓDIGO FUENTE: `private-clauseModificado3.c`

```
#include <stdio.h>

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

main()
{
```

```
    int i, n = 7;
    int a[n], suma=50;

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

    #pragma omp parallel
    {
              suma=0;

              #pragma omp for
              for (i=0; i<n; i++)
              {
                 suma = 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);
    }

    printf("\n");
    printf("suma=%d \n",suma);
}
```

CAPTURAS DE PANTALLA:

```
cvi066023:3 ivanortegaalba$ ./PrivateNoPrivate
thread 1 suma a[2] / thread 0 suma a[0] / thread 2 suma a[4] / thread 3 suma a[6
] / thread 1 suma a[3] / thread 0 suma a[1] / thread 2 suma a[5] /
* thread 3 suma= 15
* thread 0 suma= 15
* thread 1 suma= 15
* thread 2 suma= 15
suma=15
```

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:

   No, ya que se quedará con el valor de la ultima hebra en salir. En este caso es 6 ya que la ultima hebra en salir tiene este valor, pero si cambiamos el numero de hebras, probablemente este cambiará.

   CAPTURAS DE PANTALLA:

```
cvi066023:4 ivanortegaalba$ ./FirstPrivate
 thread 1 suma a[2] suma=2
 thread 3 suma a[6] suma=6
 thread 0 suma a[0] suma=0
 thread 2 suma a[4] suma=4
 thread 1 suma a[3] suma=5
 thread 0 suma a[1] suma=1
 thread 2 suma a[5] suma=9

Fuera de la construcción parallel suma=6
```

5. ¿Qué ocurre si en `copyprivate-clause.c` se elimina la cláusula `copyprivate(a)` en la directiva `single`? ¿A qué cree que es debido?

   RESPUESTA:

   Si usamos copyprivate al entrar una hebra en el bloque, copiará el valor de la variable declarada en todas las hebras en ejecución. Al eliminarla, el resto de hebras no conocerán el valor que le hemos indicado, por lo que tendrán un valor de la variable indeterminado.

   CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

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

```
#include <omp.h>

main() {

   int n = 9, i, b[n];

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

   #pragma omp parallel
   {
             int a;
//Al single solo entra una de las hebras en paralelo, y copyprivate copia el
valor que
//tome esa varaible en el resto de hebras ejecutandose en paralelo

             #pragma omp single
             {
                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");
}
```

CAPTURAS DE PANTALLA:

```
cvi066048:5 ivanortegaalba$ ./ConCopyPrivate

Introduce valor de inicialización a: 10

Single ejecutada por el thread 0
Depués de la región parallel:
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
cvi066048:5 ivanortegaalba$ ./SinCopyPrivate

Introduce valor de inicialización a: 10

Single ejecutada por el thread 1
Depués de la región parallel:
b[0] = 0        b[1] = 0        b[2] = 0        b[3] = 10       b[4] = 10       b
[5] = 0 b[6] = 0        b[7] = 32716    b[8] = 32716
cvi066048:5 ivanortegaalba$
```

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 de la acumulación más el 10 inicialmente usado en la variable. Eso quiere decir, que la variable suma solo se modifica en la acumulación y no en cada hebra. Es decir, escribe sobre la variable acumulando el valor que anteriormente tenía solo al final

del for, y no en cada hebra.

CÓDIGO FUENTE: `reduction-clauseModificado.c`

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

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

main(int argc, char **argv) {

   int i, n=20, a[n],suma=10;

   if(argc < 2) {
             fprintf(stderr,"Falta 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 for reduction(+:suma)
   for (i=0; i<n; i++)
             suma += a[i];

   printf("Tras 'parallel' suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:

```
cvi066048:6 ivanortegaalba$ ./Reduction 10
Tras 'parallel' suma=45
cvi066048:6 ivanortegaalba$ ./ModReduction 10
Tras 'parallel' suma=55
cvi066048:6 ivanortegaalba$ ./Reduction 2
Tras 'parallel' suma=1
cvi066048:6 ivanortegaalba$ ./ModReduction 2
Tras 'parallel' suma=11
```

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.

   RESPUESTA: Para hacer que escriba solo una a la vez, podemos usar la directiva `atomic`.

   CÓDIGO FUENTE: `reduction-clauseModificado7.c`

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

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

```
main(int argc, char **argv) {

   int i, n=20, a[n],suma=0;

   if(argc < 2) {
             fprintf(stderr,"Falta 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 for
   for (i=0; i<n; i++)
    #pragma omp atomic
                suma += a[i];

   printf("Tras 'parallel' suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:

```
MacBook-Pro-de-Ivan:7 ivanortegaalba$ gcc -fopenmp -O2 reduction-clause.c -o Mod
SinReduction
MacBook-Pro-de-Ivan:7 ivanortegaalba$ ./ModSinReduction 20
Tras 'parallel' suma=190
MacBook-Pro-de-Ivan:7 ivanortegaalba$ ./ModSinReduction 2
Tras 'parallel' suma=1
MacBook-Pro-de-Ivan:7 ivanortegaalba$ ./ModSinReduction 3
Tras 'parallel' suma=3
```

8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, M, por un vector, v1:

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

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).

CÓDIGO FUENTE: `pmv-secuencial.c`

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

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

```
main(int argc, char **argv) {
  int N = atoi(argv[1]);
  int i,j;
  int m[N][N];
  int v1[N],v2[N];
  double start,end,elapsed;
  if(argc < 2) {
             fprintf(stderr,"Faltan argumentos\n");
             exit(-1);
  }
  //Inicializamos
  for(i = 0; i<N;i++){
    v1[i]= i;
    v2[i] = 0;
    for(j=0;j<N;j++)
      m[i][j] = i + j;
  }

  start = omp_get_wtime();

  //Multiplicamos
  for (i = 0; i < N; ++i)
      for (j = 0; j < N; ++j)
          v2[i] += m[i][j] * v1[j];

  end = omp_get_wtime();
  elapsed = end - start;
  //Imprimimos
  printf("Vector Resultante\n");
  for(i = 0; i<N;i++)
     printf("v2[%d] = %d\n",i,v2[i]);
  printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",elapsed,N);

}
```

CAPTURAS DE PANTALLA:

```
MacBook-Pro-de-Ivan:8 ivanortegaalba$ ./PvmSecuencial 8
Vector Resultante
v2[0] = 140
v2[1] = 168
v2[2] = 196
v2[3] = 224
v2[4] = 252
v2[5] = 280
v2[6] = 308
v2[7] = 336
Tiempo(seg.):0.000000954         / Tamaño Vectores:8
MacBook-Pro-de-Ivan:8 ivanortegaalba$ ./PvmSecuencial 11
Vector Resultante
v2[0] = 385
v2[1] = 440
v2[2] = 495
v2[3] = 550
v2[4] = 605
v2[5] = 660
v2[6] = 715
v2[7] = 770
v2[8] = 825
v2[9] = 880
v2[10] = 935
Tiempo(seg.):0.000001192         / Tamaño Vectores:11
```

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).

CÓDIGO FUENTE : `pmv-OpenMP-a.c`

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

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

main(int argc, char **argv) {
  int N = atoi(argv[1]);
  int i,j;
  int m[N][N];
  int v1[N],v2[N];
  double start,end,elapsed;
  if(argc < 2) {
             fprintf(stderr,"Faltan argumentos\n");
             exit(-1);
  }
  //Inicializamos
  for(i = 0; i<N;i++){
    v1[i]= i;
    v2[i] = 0;
    for(j=0;j<N;j++)
      m[i][j] = i + j;
  }

  start = omp_get_wtime();

  //Multiplicamos
  //Declaramos j privada a cada hebra, para no pisarse en el otro bucle.
```

```
  #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];

  end = omp_get_wtime();
  elapsed = end - start;
  //Imprimimos
  printf("Vector Resultante\n");
  for(i = 0; i<N;i++)
     printf("v2[%d] = %d\n",i,v2[i]);
  printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",elapsed,N);

}
```

CÓDIGO FUENTE: pmv-OpenMP-b.c

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

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

main(int argc, char **argv) {
  int N = atoi(argv[1]);
  int i,j;
  int m[N][N];
  int v1[N],v2[N];
  double start,end,elapsed;
  if(argc < 2) {
             fprintf(stderr,"Faltan argumentos\n");
             exit(-1);
  }
  //Inicializamos
  for(i = 0; i<N;i++){
    v1[i]= i;
    v2[i] = 0;
    for(j=0;j<N;j++)
      m[i][j] = i + j;
  }

  start = omp_get_wtime();

  //Multiplicamos
  for (i = 0; i < N; ++i)
  #pragma omp parallel for
      for (j = 0; j < N; ++j)
          v2[i] += m[i][j] * v1[j];

  end = omp_get_wtime();
  elapsed = end - start;
  //Imprimimos
  printf("Vector Resultante\n");
  for(i = 0; i<N;i++)
     printf("v2[%d] = %d\n",i,v2[i]);
  printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",elapsed,N);

}
```

RESPUESTA: He tenido problemas al paralelizar por filas, ya que compartían la variable j y la incrementaban hebras distintas, por defecto es compartida por lo que no era correcta la suma. Lo he solucionado, haciéndola privada a cada hebra.

CAPTURAS DE PANTALLA:

```
MacBook-Pro-de-Ivan:9 ivanortegaalba$ ./PvmColumnas 8
Vector Resultante
v2[0] = 140
v2[1] = 168
v2[2] = 196
v2[3] = 224
v2[4] = 252
v2[5] = 280
v2[6] = 308
v2[7] = 336
Tiempo(seg.):0.000461102         / Tamaño Vectores:8
MacBook-Pro-de-Ivan:9 ivanortegaalba$ ./PvmColumnas 11
Vector Resultante
v2[0] = 385
v2[1] = 440
v2[2] = 495
v2[3] = 550
v2[4] = 605
v2[5] = 660
v2[6] = 715
v2[7] = 770
v2[8] = 825
v2[9] = 880
v2[10] = 935
Tiempo(seg.):0.000494003         / Tamaño Vectores:11
```

```
MacBook-Pro-de-Ivan:9 ivanortegaalba$ ./PvmFilas 8
Vector Resultante
v2[0] = 140
v2[1] = 168
v2[2] = 196
v2[3] = 224
v2[4] = 252
v2[5] = 280
v2[6] = 308
v2[7] = 336
Tiempo(seg.):0.000159025         / Tamaño Vectores:8
MacBook-Pro-de-Ivan:9 ivanortegaalba$ ./PvmFilas 11
Vector Resultante
v2[0] = 385
v2[1] = 440
v2[2] = 495
v2[3] = 550
v2[4] = 605
v2[5] = 660
v2[6] = 715
v2[7] = 770
v2[8] = 825
v2[9] = 880
v2[10] = 935
Tiempo(seg.):0.000151873         / Tamaño Vectores:11
```

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).

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

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

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

main(int argc, char **argv) {
  int N = atoi(argv[1]);
  int i,j;
  int m[N][N];
  int v1[N],v2[N];
  double start,end,elapsed;
  if(argc < 2) {
             fprintf(stderr,"Faltan argumentos\n");
             exit(-1);
  }
  //Inicializamos
  for(i = 0; i<N;i++){
    v1[i]= i;
    v2[i] = 0;
    for(j=0;j<N;j++)
      m[i][j] = i + j;
  }

  start = omp_get_wtime();
  int suma=0;
  //Multiplicamos

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


  end = omp_get_wtime();
  elapsed = end - start;
  //Imprimimos
  printf("Vector Resultante\n");
  for(i = 0; i<N;i++)
     printf("v2[%d] = %d\n",i,v2[i]);
  printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",elapsed,N);

}
```

RESPUESTA: No he tenido muchos problemas para subsanarlo.

CAPTURAS DE PANTALLA:

```
MacBook-Pro-de-Ivan:10 ivanortegaalba$ ./PvmReduction 8
Vector Resultante
v2[0] = 140
v2[1] = 168
v2[2] = 196
v2[3] = 224
v2[4] = 252
v2[5] = 280
v2[6] = 308
v2[7] = 336
Tiempo(seg.):0.000471830         / Tamaño Vectores:8
MacBook-Pro-de-Ivan:10 ivanortegaalba$ ./PvmReduction 11
Vector Resultante
v2[0] = 385
v2[1] = 440
v2[2] = 495
v2[3] = 550
v2[4] = 605
v2[5] = 660
v2[6] = 715
v2[7] = 770
v2[8] = 825
v2[9] = 880
v2[10] = 935
Tiempo(seg.):0.000576973         / Tamaño Vectores:11
```

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 el PC del aula de prácticas de los tres códigos implementados en los ejercicios anteriores para tres tamaños (N) distintos (consulte la Lección 6/Tema 2). Recuerde usar –O2 al compilar.

    TABLA Y GRÁFICA (por *ejemplo* para 1-4 hebras PC aula, y para 1-12 hebras en atcgrid, tamaños-N-: 1.000, 10.000, 100.000):

    COMENTARIOS SOBRE LOS RESULTADOS: