Grai2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

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

Estudiante (nombre y apellidos): Jose Teodosio Lorente Vallecillos
Grupo de prácticas y profesor de prácticas: A3 Mancia
Fecha de entrega:
Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Ejercicios basados en los ejemplos del seminario práctico

1. **(a)** Añadir la cláusula `default(none)` a la directiva `parallel` del ejemplo del seminario `shared-clause.c`? ¿Qué ocurre? ¿A qué se debe? **(b)** Resolver el problema generado sin eliminar `default(none)`. Incorporar el código con la modificación al cuaderno de prácticas. (Añadir capturas de pantalla que muestren lo que ocurre)

RESPUESTA: Cuando se le añade la clausula default(none) a la directiva parallel ocurre un error, exactamente un error referido a la variable n la cual dice que no esta declarada.

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer1] 2021-04-14 miércoles
$gcc -fopenmp -O2 -o shared-clauseModificado shared-clauseModificado.c
shared-clauseModificado.c: In function ‘main’:
shared-clauseModificado.c:14:11: error: ‘n’ not specified in enclosing ‘parallel’
   14 |   #pragma omp parallel for default(none) shared(a)
      |           ^~~
shared-clauseModificado.c:14:11: error: enclosing ‘parallel’
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer1] 2021-04-14 miércoles
$
```

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

```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 default(none) shared(a,n)
    for(i=0; i<n; i++)
      a[i] += i;

    printf("Despues de parallel for:\n");

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

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer1] 2021-04-14 miércoles
$gcc -fopenmp -O2 -o shared-clauseModificado shared-clauseModificado.c
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer1] 2021-04-14 miércoles
$./shared-clauseModificado
Despues de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer1] 2021-04-14 miércoles
$
```

2. **(a)** Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel`. Inicializar `suma` dentro del `parallel` a un valor distinto de 0. Ejecutar varias veces el código ¿Qué imprime el código fuera del `parallel`? (mostrar lo que ocurre con una captura de pantalla) Razonar respuesta. **(b)** Modificar el código del apartado (a) para que se inicialice `suma` fuera del `parallel` en lugar de dentro ¿Qué ocurre? Comparar todo lo que imprime el código ahora con la salida en (a) (mostrar la salida con una captura de pantalla) Razonar respuesta.

(a) RESPUESTA:

El código fuera del parallel imprime suma=0. Esto es debido a la privacidad de la suma por el uso de private, cada thread realiza su suma privada saliendo del parallel sin guardar el resultado en la variable suma original.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_a.c`

```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 private(suma)
  {
    suma=3;
    #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");
}
```

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer2] 2021-04-26 lunes
$./private-clauseModificado_a
thread 2 suma a[5]/thread 2 suma a[6]/thread 0 suma a[0]/thread 0 suma a[1]/thread 0 suma a[2]/thread 1 s
uma a[3]/thread 1 suma a[4]/
* thread 0 suma= 0
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer2] 2021-04-26 lunes
$./private-clauseModificado_a
thread 0 suma a[0]/thread 0 suma a[1]/thread 0 suma a[2]/thread 2 suma a[5]/thread 2 suma a[6]/thread 1 s
uma a[3]/thread 1 suma a[4]/
* thread 0 suma= 0
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer2] 2021-04-26 lunes
$
```

(b) RESPUESTA:
Si se inicia la variable suma fuera del parallel sigue sin guardarse el resultado en la variable suma original e imprime el valor con la que lo hemos inicializado.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_b.c`

```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=3;

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

  #pragma omp parallel private(suma)
  {
    //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");
}
```

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer2] 2021-04-26 lunes
$gcc -O2 -fopenmp -o private-clauseModificado_b private-clauseModificado_b.c
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer2] 2021-04-26 lunes
$./private-clauseModificado_b
thread 2 suma a[5]/thread 2 suma a[6]/thread 1 suma a[3]/thread 1 suma a[4]/thread 0 suma a[0]/thread 0 s
uma a[1]/thread 0 suma a[2]/
* thread 0 suma= 3
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer2] 2021-04-26 lunes
$./private-clauseModificado_b
thread 0 suma a[0]/thread 0 suma a[1]/thread 0 suma a[2]/thread 1 suma a[3]/thread 1 suma a[4]/thread 2 s
uma a[5]/thread 2 suma a[6]/
* thread 0 suma= 3
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer2] 2021-04-26 lunes
$
```

3. **(a)** Eliminar la cláusula `private(suma)` en `private-clause.c`. Ejecutar el código resultante. ¿Qué ocurre? **(b)** ¿A qué es debido?

RESPUESTA:

Suma será una variable compartida para cada thread y como consecuencia obtendrá un resultado erróneo del ultimo thread que escribiera la variable.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

```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 = 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");
}
```

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer3] 2021-04-26 lunes
$./private-clauseModificado3
thread 1 suma a[1]/thread 4 suma a[4]/thread 6 suma a[6]/thread 2 suma a[2]/thread 3 suma a[3]/thread 0 s
uma a[0]/thread 5 suma a[5]/
* thread 7 suma= 6
* thread 6 suma= 6
* thread 3 suma= 6
* thread 5 suma= 6
* thread 0 suma= 6
* thread 4 suma= 6
* thread 2 suma= 6
* thread 1 suma= 6
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer3] 2021-04-26 lunes
$./private-clauseModificado3
thread 6 suma a[6]/thread 1 suma a[1]/thread 2 suma a[2]/thread 4 suma a[4]/thread 3 suma a[3]/thread 5 s
uma a[5]/thread 0 suma a[0]/
* thread 7 suma= 0
* thread 2 suma= 0
* thread 0 suma= 0
* thread 1 suma= 0
* thread 4 suma= 0
* thread 6 suma= 0
* thread 3 suma= 0
* thread 5 suma= 0
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer3] 2021-04-26 lunes
$./private-clauseModificado3
thread 2 suma a[2]/thread 1 suma a[1]/thread 6 suma a[6]/thread 3 suma a[3]/thread 4 suma a[4]/thread 5 s
uma a[5]/thread 0 suma a[0]/
* thread 7 suma= 6
* thread 1 suma= 6
* thread 4 suma= 6
* thread 5 suma= 6
* thread 6 suma= 6
* thread 3 suma= 6
* thread 2 suma= 6
* thread 0 suma= 6
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer3] 2021-04-26 lunes
$./private-clauseModificado3
thread 1 suma a[1]/thread 5 suma a[5]/thread 2 suma a[2]/thread 3 suma a[3]/thread 4 suma a[4]/thread 6 s
uma a[6]/thread 0 suma a[0]/
* thread 7 suma= 5
* thread 6 suma= 5
* thread 5 suma= 5
* thread 2 suma= 5
* thread 4 suma= 5
* thread 3 suma= 5
* thread 0 suma= 5
* thread 1 suma= 5
```

4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 6 fuera de la región `parallel`. **(a)** Cambiar el tamaño del vector a 10. Razonar lo que imprime el código en su PC con esta modificación. (añadir capturas de pantalla que muestren lo que ocurre). **(b)** Sin cambiar el tamaño del vector ¿podría imprimir el código otro valor? Razonar respuesta (añadir capturas de pantalla que muestren lo que ocurre).

(a) RESPUESTA:

Ahora el código siempre imprime 9 debido a la directiva lastprivate, la cual copia al salir del parallel el ultimo valor de una ejecución secuencial.

CAPTURAS DE PANTALLA:

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

int main(){

  int i,n=10;
  int a[n],suma=0;

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

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

    printf("\nFuera de la construccion parallel suma=%d\n",suma);
}
```

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer4] 2021-04-26 lunes
$gcc -O2 -fopenmp -o firstlastprivate_a firstlastprivate_a.c
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer4] 2021-04-26 lunes
$./firstlastprivate_a
 thread 3 suma a[5] suma=5
 thread 7 suma a[9] suma=9
 thread 4 suma a[6] suma=6
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 5 suma a[7] suma=7
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 6 suma a[8] suma=8
 thread 2 suma a[4] suma=4

Fuera de la construccion parallel suma=9
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer4] 2021-04-26 lunes
$./firstlastprivate_a
 thread 2 suma a[4] suma=4
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 6 suma a[8] suma=8
 thread 5 suma a[7] suma=7
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 3 suma a[5] suma=5
 thread 4 suma a[6] suma=6
 thread 7 suma a[9] suma=9

Fuera de la construccion parallel suma=9
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer4] 2021-04-26 lunes
$
```

(b) RESPUESTA:
Si, sin la directiva lastprivate el valor que imprime ya no seria 9 sino 0 ya que no se guarda el valor de la variable suma de manera compartida.
CAPTURAS DE PANTALLA:

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

int main(){

  int i,n=10;
  int a[n],suma=0;

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

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

    printf("\nFuera de la construccion parallel suma=%d\n",suma);
}
```

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer4] 2021-04-26 lunes
$gcc -O2 -fopenmp -o firstlastprivate_b firstlastprivate_b.c
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer4] 2021-04-26 lunes
$./firstlastprivate_b
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 6 suma a[8] suma=8
 thread 2 suma a[4] suma=4
 thread 4 suma a[6] suma=6
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 7 suma a[9] suma=9
 thread 3 suma a[5] suma=5
 thread 5 suma a[7] suma=7

Fuera de la construccion parallel suma=0
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer4] 2021-04-26 lunes
$./firstlastprivate_b
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 2 suma a[4] suma=4
 thread 4 suma a[6] suma=6
 thread 7 suma a[9] suma=9
 thread 5 suma a[7] suma=7
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 6 suma a[8] suma=8
 thread 3 suma a[5] suma=5

Fuera de la construccion parallel suma=0
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer4] 2021-04-26 lunes
$./firstlastprivate_b
 thread 2 suma a[4] suma=4
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 4 suma a[6] suma=6
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 7 suma a[9] suma=9
 thread 5 suma a[7] suma=7
 thread 3 suma a[5] suma=5
 thread 6 suma a[8] suma=8

Fuera de la construccion parallel suma=0
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer4] 2021-04-26 lunes
$
```

5. **(a)** ¿Qué se observa en los resultados de ejecución de `copyprivate-clause.c` cuando se elimina la cláusula `copyprivate(a)` en la directiva `single`? **(b)** ¿A qué cree que es debido? (añadir una captura de pantalla que muestre lo que ocurre)

RESPUESTA:

Al eliminar copyprivate el valor de A es distinto en casi todos los threads, copyprivate realiza copias de una variable privada a cada una de las hebras con ese mismo nombre. Por eso si lo eliminas no todas las A de las hebras tendrán ese valor dado.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```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 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");
}
```

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer5] 2021-04-26 lunes
$gcc -O2 -fopenmp -o copyprivate-clauseModificado copyprivate-clauseModificado.c
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer5] 2021-04-26 lunes
$./copyprivate-clauseModificado

Introduce valor de inicializacion a: 10

Single ejecutada por el thread 1
Despues de la region parallel:
b[0] = 21900     b[1] = 21900    b[2] = 10       b[3] = 0        b[4] = 0        b[5] = 0        b[6] = 0
 b[7] = 0        b[8] = 0
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer5] 2021-04-26 lunes
$./copyprivate-clauseModificado

Introduce valor de inicializacion a: 10

Single ejecutada por el thread 1
Despues de la region parallel:
b[0] = 21962     b[1] = 21962    b[2] = 10       b[3] = 0        b[4] = 0        b[5] = 0        b[6] = 0
 b[7] = 0        b[8] = 0
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer5] 2021-04-26 lunes
$
```

6. En el ejemplo `reduction-clause.c` sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado (añada capturas de pantalla que muestren lo que ocurre)

RESPUESTA:

Imprime el resultado aumentando su valor en diez debido a que hemos inicializado la variable suma a diez y con reduction no es necesario inicializar dentro de la directiva, para y por eso en reduction el operador reducción en la suma su valor inicial debería ser cero.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

```c
#include <stdio.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;

  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:

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer6] 2021-04-26 lunes
$./reduction-clauseModificado 4
Tras 'parallel' suma=16
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer6] 2021-04-26 lunes
$./reduction-clauseModificado 6
Tras 'parallel' suma=25
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer6] 2021-04-26 lunes
$./reduction-clauseModificado 8
Tras 'parallel' suma=38
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer6] 2021-04-26 lunes
$./reduction-clauseModificado 10
Tras 'parallel' suma=55
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer6] 2021-04-26 lunes
$./reduction-clause 4
Tras 'parallel' suma=6
```

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 añadir más directivas de trabajo compartido (añada capturas de pantalla que muestren lo que ocurre).

RESPUESTA: Tras eliminar reduction, añadiendo la directiva atomic mediante la que se accede a una ubicación de almacenamiento en especifico pudiendo así los distintos threads escribir en esta sin dar valores indefinidos se sigue realizando la suma sin problema.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

```c
#include <stdio.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=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:

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer7] 2021-04-26 lunes
$gcc -O2 -fopenmp -o reduction-clauseModificado7 reduction-clauseModificado7.c
reduction-clauseModificado7.c: In function ‘main’:
reduction-clauseModificado7.c:13:5: warning: implicit declaration of function ‘exit’ [-Wimplicit-function
-declaration]
   13 |     exit(-1);
      |     ^~~~
reduction-clauseModificado7.c:13:5: warning: incompatible implicit declaration of built-in function ‘exit
’
reduction-clauseModificado7.c:4:1: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
    3 |   #include <omp.h>
  +++ |+#include <stdlib.h>
    4 | #else
reduction-clauseModificado7.c:16:6: warning: implicit declaration of function ‘atoi’ [-Wimplicit-function
-declaration]
   16 |   n= atoi(argv[1]);
      |      ^~~~
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer7] 2021-04-26 lunes
$./reduction-clauseModificado7 4
Tras 'parallel' suma=6
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer7] 2021-04-26 lunes
$./reduction-clauseModificado7 6
Tras 'parallel' suma=15
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer7] 2021-04-26 lunes
$./reduction-clauseModificado7 8
Tras 'parallel' suma=28
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer7] 2021-04-26 lunes
$./reduction-clauseModificado7 10
Tras 'parallel' suma=45
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp2/ejer7] 2021-04-26 lunes
$
```

Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

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`

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <malloc.h>

#define GLOBAL

#ifdef GLOBAL
  #define MAX 67108864 //2^26
#endif

int main(int argc, char const *argv[]){

  if(argc != 2){
    printf("Error introduccion argumentos %s", argv[0]);
    exit(-1);
  }

  struct timespec time1, time2;
  double timeResult;
  int N = atoi(argv[1]);

  #ifdef GLOBAL
    if(N > MAX)
      N=MAX;

    int matrix[N][N];
    int vector[N];
    int vectorResult[N];
    printf(" GLOBAL \n");
  #endif

  for(int i=0; i<N; i++){
    vector[i]=i;
    for(int j=0; j<N;j++)
      matrix[i][j]=i+j;
  }

  //Medicion del tiempo
  clock_gettime(CLOCK_REALTIME, &time1);
  //Calculo del vector resultante
  for(int i=0; i<N; i++){
    int suma=0;
    for(int j=0; j<N; j++){
      suma+=matrix[i][j]*vector[j];
    }

    vectorResult[i]=suma;
  }
  clock_gettime(CLOCK_REALTIME, &time2);
  timeResult= (double) (time2.tv_sec-time1.tv_sec) + (double) (time2.tv_nsec-time1.tv_nsec) / (1.e+9);
```

```
  printf("Tiempo: %11.9f\t / Tamanio vectores: %u\n",timeResult,N);
  if(N<15)
    for(int i=0; i<N; i++){
      printf("VECTORRESULT[%d] = %d \n",i,vectorResult[i]);
    }
  else{
    printf("VECTORRESULT[0] = %d ",vectorResult[0]);
    printf("VECTORRESULT[%d] = %d \n",N-1,vectorResult[N-1]);
  }

}
```

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

#define DINAMIC

int main(int argc, char const *argv[]){

  if(argc != 2){
    printf("Error introduccion argumentos %s", argv[0]);
    exit(-1);
  }

  struct timespec time1, time2;
  double timeResult;
  int N = atoi(argv[1]);

  #ifdef DINAMIC
    int **matrix;
    int *vector;
    int *vectorResult;
    matrix = (int**) malloc(N * sizeof(int*));
    for(int i=0; i<N; i++)
      matrix[i]=(int*) malloc(N * sizeof(int));

    vector=(int*) malloc(N * sizeof(int));
    vectorResult=(int*) malloc(N * sizeof(int));
    printf(" DINAMIC \n");
  #endif

  for(int i=0; i<N; i++){
    vector[i]=i;
    for(int j=0; j<N;j++)
      matrix[i][j]=i+j;
  }

  //Medicion del tiempo
  clock_gettime(CLOCK_REALTIME, &time1);
  //Calculo del vector resultante
  for(int i=0; i<N; i++){
    int suma=0;
    for(int j=0; j<N; j++){
      suma+=matrix[i][j]*vector[j];
    }

    vectorResult[i]=suma;
  }
  clock_gettime(CLOCK_REALTIME, &time2);
  timeResult= (double) (time2.tv_sec-time1.tv_sec) + (double) (time2.tv_nsec-time1.tv_nsec) / (1.e+9);
```

```
  printf("Tiempo: %11.9f\t / Tamanio vectores: %u\n",timeResult,N);
  if(N<15)
    for(int i=0; i<N; i++){
      printf("VECTORRESULT[%d] = %d \n",i,vectorResult[i]);
    }
  else{
    printf("VECTORRESULT[0] = %d ",vectorResult[0]);
    printf("VECTORRESULT[%d] = %d \n",N-1,vectorResult[N-1]);
  }

  #ifdef DINAMIC
    for(int i=0; i<N; i++)
      free(matrix[i]);

    free(matrix);
    free(vector);
    free(vectorResult);
  #endif
}
```

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp2/ejer8] 2021-04-27 martes
$srun ./pmv-secuencialdinamic 10
 DINAMIC
Tiempo: 0.000000390      / Tamanio vectores: 10
VECTORRESULT[0] = 285
VECTORRESULT[1] = 330
VECTORRESULT[2] = 375
VECTORRESULT[3] = 420
VECTORRESULT[4] = 465
VECTORRESULT[5] = 510
VECTORRESULT[6] = 555
VECTORRESULT[7] = 600
VECTORRESULT[8] = 645
VECTORRESULT[9] = 690
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp2/ejer8] 2021-04-27 martes
$srun ./pmv-secuencialglobal 10
 GLOBAL
Tiempo: 0.000000352      / Tamanio vectores: 10
VECTORRESULT[0] = 285
VECTORRESULT[1] = 330
VECTORRESULT[2] = 375
VECTORRESULT[3] = 420
VECTORRESULT[4] = 465
VECTORRESULT[5] = 510
VECTORRESULT[6] = 555
VECTORRESULT[7] = 600
VECTORRESULT[8] = 645
VECTORRESULT[9] = 690
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp2/ejer8] 2021-04-27 martes
$
```

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).

Realizado con la versión dinámica

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

```
#pragma omp parallel for
for(int i=0; i<N; i++){
  vector[i]=i;
  #pragma omp parallel for
  for(int j=0; j<N;j++)
    matrix[i][j]=i+j;
}

//Medicion del tiempo
clock_gettime(CLOCK_REALTIME, &time1);
//Calculo del vector resultante
#pragma omp parallel for
for(int i=0; i<N; i++){
  int suma=0;
  for(int j=0; j<N; j++){
    suma+=matrix[i][j]*vector[j];
  }

  vectorResult[i]=suma;
}
```

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

```
#pragma omp parallel for
for(int i=0; i<N; i++){
  vector[i]=i;
  #pragma omp parallel for
  for(int j=0; j<N;j++)
    matrix[i][j]=i+j;
}

//Medicion del tiempo
clock_gettime(CLOCK_REALTIME, &time1);
//Calculo del vector resultante
for(int i=0; i<N; i++){
  int suma=0;
  #pragma omp parallel for
  for(int j=0; j<N; j++){
    #pragma omp atomic
    suma+=matrix[i][j]*vector[j];
  }

  vectorResult[i]=suma;
}
```

RESPUESTA:
En el código a se ha insertado la directiva parallel para la paralelización del bucle que recorre la matriz su inicialización numero de veces; y en el código b con un método similar salvo con la directiva atomic para el

acceso de las hebras a una localización de memoria automáticamente.
No me ha generado ningún error el código tanto en compilación como en ejecución.
CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp2/ejer9] 2021-04-27 martes
$./pmv-OpenMP-a 8
 DINAMIC
Tiempo: 0.000002279      / Tamanio vectores: 8
VECTORRESULT[0] = 140
VECTORRESULT[1] = 168
VECTORRESULT[2] = 196
VECTORRESULT[3] = 224
VECTORRESULT[4] = 252
VECTORRESULT[5] = 280
VECTORRESULT[6] = 308
VECTORRESULT[7] = 336
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp2/ejer9] 2021-04-27 martes
$./pmv-OpenMP-a 11
 DINAMIC
Tiempo: 0.000005205      / Tamanio vectores: 11
VECTORRESULT[0] = 385
VECTORRESULT[1] = 440
VECTORRESULT[2] = 495
VECTORRESULT[3] = 550
VECTORRESULT[4] = 605
VECTORRESULT[5] = 660
VECTORRESULT[6] = 715
VECTORRESULT[7] = 770
VECTORRESULT[8] = 825
VECTORRESULT[9] = 880
VECTORRESULT[10] = 935
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp2/ejer9] 2021-04-27 martes
$./pmv-OpenMP-b 8
 DINAMIC
Tiempo: 0.000033732      / Tamanio vectores: 8
VECTORRESULT[0] = 140
VECTORRESULT[1] = 168
VECTORRESULT[2] = 196
VECTORRESULT[3] = 224
VECTORRESULT[4] = 252
VECTORRESULT[5] = 280
VECTORRESULT[6] = 308
VECTORRESULT[7] = 336
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp2/ejer9] 2021-04-27 martes
$./pmv-OpenMP-b 11
 DINAMIC
Tiempo: 0.000049272      / Tamanio vectores: 11
VECTORRESULT[0] = 385
VECTORRESULT[1] = 440
VECTORRESULT[2] = 495
VECTORRESULT[3] = 550
VECTORRESULT[4] = 605
VECTORRESULT[5] = 660
VECTORRESULT[6] = 715
VECTORRESULT[7] = 770
VECTORRESULT[8] = 825
VECTORRESULT[9] = 880
VECTORRESULT[10] = 935
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp2/ejer9] 2021-04-27 martes
$
```

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`

```
#pragma omp parallel for
for(int i=0; i<N; i++){
  vector[i]=i;
  #pragma omp parallel for
  for(int j=0; j<N;j++)
    matrix[i][j]=i+j;
}

//Medicion del tiempo
clock_gettime(CLOCK_REALTIME, &time1);
//Calculo del vector resultante
for(int i=0; i<N; i++){
  int suma=0;
  #pragma omp parallel for reduction(+:suma)
  for(int j=0; j<N; j++){
    suma+=matrix[i][j]*vector[j];
  }

  vectorResult[i]=suma;
}
```

RESPUESTA:
No me ha generado ningún error el código tanto en compilación como en ejecución.
CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp2/ejer10] 2021-04-27 martes
$srun ./pmv-OpenMP-reduction 8
 DINAMIC
Tiempo: 0.000018932      / Tamanio vectores: 8
VECTORRESULT[0] = 140
VECTORRESULT[1] = 168
VECTORRESULT[2] = 196
VECTORRESULT[3] = 224
VECTORRESULT[4] = 252
VECTORRESULT[5] = 280
VECTORRESULT[6] = 308
VECTORRESULT[7] = 336
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp2/ejer10] 2021-04-27 martes
$srun ./pmv-OpenMP-reduction 11
 DINAMIC
Tiempo: 0.000025224      / Tamanio vectores: 11
VECTORRESULT[0] = 385
VECTORRESULT[1] = 440
VECTORRESULT[2] = 495
VECTORRESULT[3] = 550
VECTORRESULT[4] = 605
VECTORRESULT[5] = 660
VECTORRESULT[6] = 715
VECTORRESULT[7] = 770
VECTORRESULT[8] = 825
VECTORRESULT[9] = 880
VECTORRESULT[10] = 935
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp2/ejer10] 2021-04-27 martes
$
```

11. Realizar una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid4, en uno de los nodos de la cola ac 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):

JUSTIFICAR AHORA EN BASE AL CÓDIGO LA DIFERENCIA EN TIEMPOS:

CAPTURA DE PANTALLA del script `pmv-OpenmMP-script.sh`

CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):

TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia):

Tabla 1. Tiempos de ejecución del código secuencial y de la versión paralela para atcgrid y para el PC personal

	atcgrid1, atcgrid2 o atcgrid3				**atcgrid4**				**PC**			
	Tamaño= entre 5000 y 10000		**Tamaño= entre 10000 y 100000**		**Tamaño= entre 5000 y 10000**		**Tamaño= entre 10000 y 100000**		**Tamaño= entre 5000 y 10000**		**Tamaño= entre 10000 y 100000**	
Nº de núcleos (p)	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**
Código Secuencial		----		----		----		----		----		----
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
32												

COMENTARIOS SOBRE LOS RESULTADOS: