

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Adrián Portillo Sánchez

Grupo de prácticas: A2

Fecha de entrega:

Fecha evaluación en clase:

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:

Se produce un error en el que la variable `n` se detecta como una variable que no existe dentro del `parallel`, ya que para las variables usadas debe ser especificado su alcance en la construcción.

CÓDIGO FUENTE: `shared-clauseModificado.c`

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

main(){

int i,n=7;
int a[n];

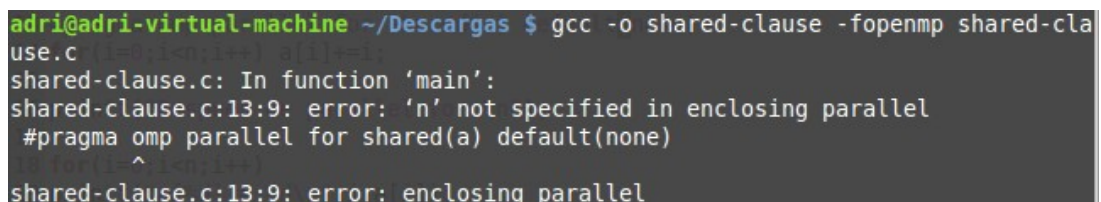
for(i=0;i<n;i++)
a[i]=i+1;
#pragma omp parallel for shared(a, n) default(none)
for(i=0;i<n;i++) a[i]+=i;

printf("Después de parallel for:\n");

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

}
```

CAPTURAS DE PANTALLA:



```
adri@adri-virtual-machine ~/Descargas $ gcc -o shared-clause -fopenmp shared-cla
use.c
shared-clause.c: In function 'main':
shared-clause.c:13:9: error: 'n' not specified in enclosing parallel
  #pragma omp parallel for shared(a) default(none)
                        ^
shared-clause.c:13:9: error: enclosing parallel
```

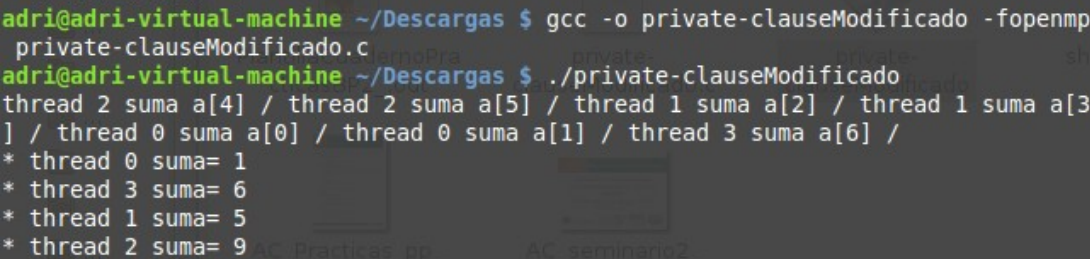
2. ¿Qué ocurre si en `private-clause.c` se inicializa la variable `suma` fuera de la construcción `parallel` en lugar de dentro? Razone su respuesta. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA:

No varía el resultado de la ejecución, ya que al llevar la cláusula `private(suma)`, cada una de las hebras lleva su propia copia de la variable por su cuenta, por otro lado, si no la llevará, el resultado sería la suma de las variables `suma` de todas las hebras en lugar del resultado deseado.

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;
    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");
}
```

CAPTURAS DE PANTALLA:


```
adri@adri-virtual-machine ~/Descargas $ gcc -o private-clauseModificado -fopenmp
private-clauseModificado.c
adri@adri-virtual-machine ~/Descargas $ ./private-clauseModificado
thread 2 suma a[4] / thread 2 suma a[5] / thread 1 suma a[2] / thread 1 suma a[3]
] / thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] /
* thread 0 suma= 1
* thread 3 suma= 6
* thread 1 suma= 5
* thread 2 suma= 9
```

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

RESPUESTA: Que el resultado será el resultado de la suma de uno de los vectores en las 4 hebras, el último en acabar, ya que cada hebra no tendrá una copia privada sino que modificará la misma variable suma.

CÓDIGO FUENTE: private-clauseModificado2.c

```
...
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:

```
adri@adri-virtual-machine ~/Descargas $ ./private-clauseModificado2
thread 2 suma a[4] / thread 2 suma a[5] / thread 1 suma a[2] / thread 1 suma a[3]
] / thread 3 suma a[6] / thread 0 suma a[0] / thread 0 suma a[1] /
* thread 0 suma= 1
* thread 1 suma= 1
* thread 2 suma= 1
* thread 3 suma= 1
adri@adri-virtual-machine ~/Descargas $ ./private-clauseModificado2
thread 0 suma a[0] / thread 0 suma a[1] / thread 1 suma a[2] / thread 1 suma a[3]
] / thread 2 suma a[4] / thread 2 suma a[5] / thread 3 suma a[6] /
* thread 0 suma= 6
* thread 1 suma= 6
* thread 2 suma= 6
* thread 3 suma= 6
adri@adri-virtual-machine ~/Descargas $ ./private-clauseModificado2
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 a[0] / thread 0 suma a[1] /
* thread 1 suma= 11
* thread 3 suma= 11
* thread 2 suma= 11
* thread 0 suma= 11
```

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 imprime la copia del último valor en una ejecución secuencial de suma, por lo que el resultado que imprime siempre será 6.

CAPTURAS DE PANTALLA:

```
thread 0 suma a[1] suma=1
Fuera de la construcción parallel suma=6
adri@adri-virtual-machine ~/Descargas $ ./firstlastprivate
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 3 suma a[6] suma=6
Fuera de la construcción parallel suma=6
adri@adri-virtual-machine ~/Descargas $ ./firstlastprivate
thread 0 suma a[0] suma=0
thread 3 suma a[6] suma=6
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 2 suma a[4] suma=4
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:

Que en lugar de inicializarse todos los elementos del vector al mismo valor, se inicializan a otros valores distintos, esto es así porque la variable creada no se difunde a las otras hebras, por lo que esas hebras toman valores basura alojados en la memoria, en lugar de tomar el valor que se le ha dado.

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;

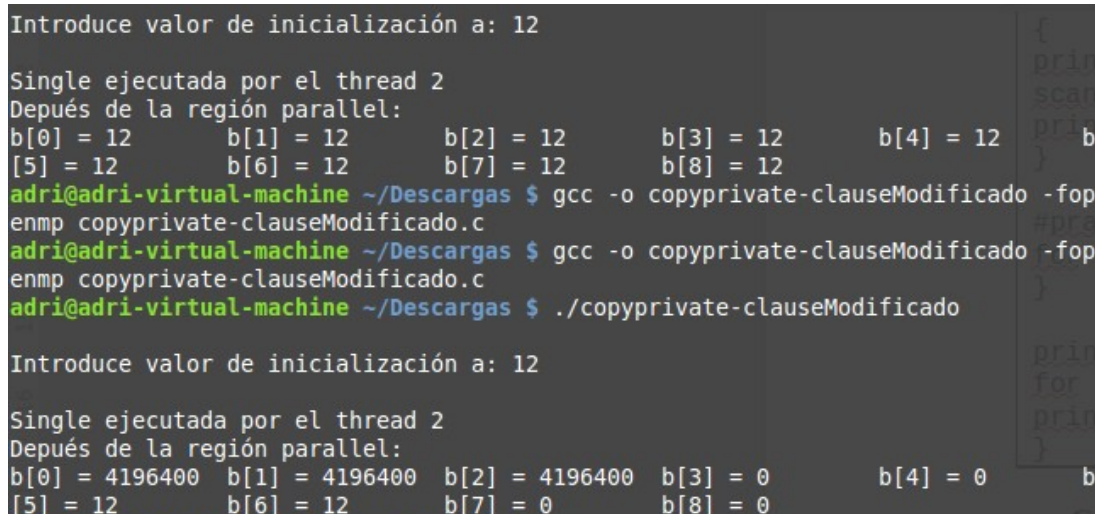
    #pragma omp parallel
    {
        int a;

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

CAPTURAS DE PANTALLA:



```
Introduce valor de inicialización a: 12

Single ejecutada por el thread 2
Después de la región parallel:
b[0] = 12      b[1] = 12      b[2] = 12      b[3] = 12      b[4] = 12
[5] = 12      b[6] = 12      b[7] = 12      b[8] = 12
adri@adri-virtual-machine ~/Descargas $ gcc -o copyprivate-clauseModificado -fopenmp copyprivate-clauseModificado.c
adri@adri-virtual-machine ~/Descargas $ gcc -o copyprivate-clauseModificado -fopenmp copyprivate-clauseModificado.c
adri@adri-virtual-machine ~/Descargas $ ./copyprivate-clauseModificado

Introduce valor de inicialización a: 12

Single ejecutada por el thread 2
Después de la región parallel:
b[0] = 4196400 b[1] = 4196400 b[2] = 4196400 b[3] = 0      b[4] = 0      b
[5] = 12      b[6] = 12      b[7] = 0      b[8] = 0
```

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

RESPUESTA:

Que en lugar de imprimir el valor de la suma de los valores imprimirá ese valor +10, ya que al inicializar la variable a 10 y comenzar a sumar a partir de ahí conseguimos el resultado final correcto +10.

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;

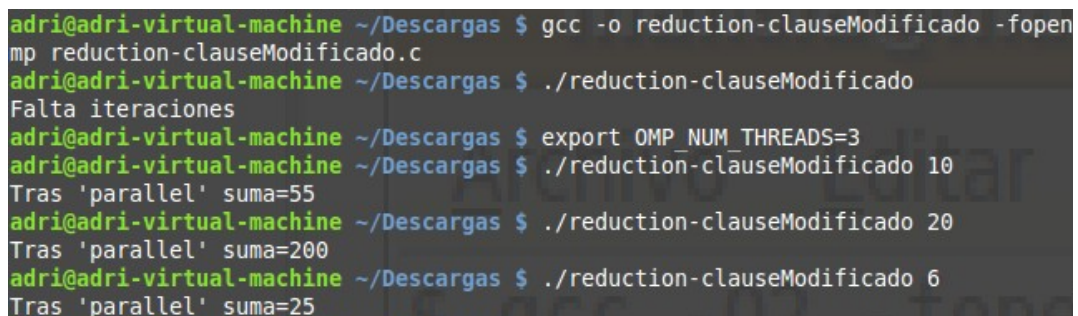
    if(argc < 2) {
        fprintf(stderr, "Falta iteraciones\n");
        exit(-1);
    }

    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:


```
adri@adri-virtual-machine ~/Descargas $ gcc -o reduction-clauseModificado -fopenmp reduction-clauseModificado.c
adri@adri-virtual-machine ~/Descargas $ ./reduction-clauseModificado
Falta iteraciones
adri@adri-virtual-machine ~/Descargas $ export OMP_NUM_THREADS=3
adri@adri-virtual-machine ~/Descargas $ ./reduction-clauseModificado 10
Tras 'parallel' suma=55
adri@adri-virtual-machine ~/Descargas $ ./reduction-clauseModificado 20
Tras 'parallel' suma=200
adri@adri-virtual-machine ~/Descargas $ ./reduction-clauseModificado 6
Tras 'parallel' suma=25
```


7. En el ejemplo `reduction-clause.c`, elimine `for` 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:

Poniendo la variable `i` como `shared`, y en cada hebra ejecutando la suma de `a[i]` tantas veces como sea necesario mientras `i` sea menor que `n`, sumando uno a `i` cada vez, se puede realizar esto, que es como simular un `for`.

CÓDIGO FUENTE: `reduction-clauseModificado2.c`

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

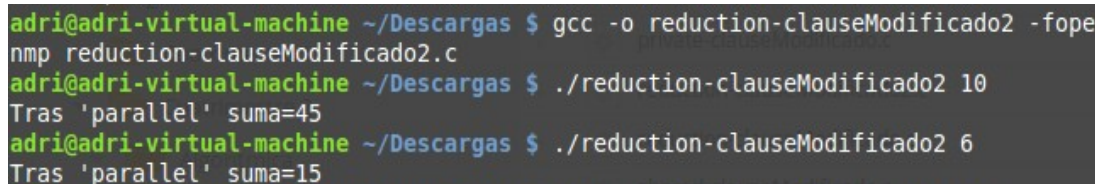
main(int argc, char **argv) {
    int i, n=20, a[n], suma=0;

    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;

    i=0;

    #pragma omp parallel shared(i) reduction(+:suma)
    {
        while(i<n){
            suma += a[i];
            i++;
        }
        printf("Tras 'parallel' suma=%d\n", suma);
    }
}
```

CAPTURAS DE PANTALLA:


```
adri@adri-virtual-machine ~/Descargas $ gcc -o reduction-clauseModificado2 -fopenmp reduction-clauseModificado2.c
adri@adri-virtual-machine ~/Descargas $ ./reduction-clauseModificado2 10
Tras 'parallel' suma=45
adri@adri-virtual-machine ~/Descargas $ ./reduction-clauseModificado2 6
Tras 'parallel' suma=15
```

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

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

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).

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 i, j, filas, columnas, v[100], M[100][100], s[100];

    if(argc < 3) {
        fprintf(stderr, "Falta filas o columnas\n");
        exit(-1);
    }

    filas = atoi(argv[1]);
    columnas = atoi(argv[2]);

    for(i=0; i<filas; i++){
        v[i]=i;
        s[i]=0;
    }

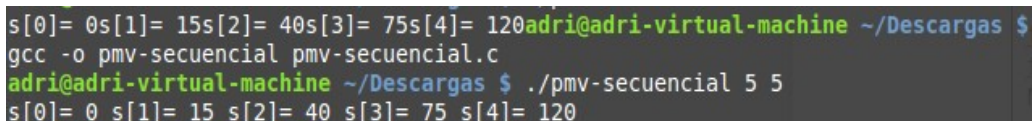
    for(i=0; i<filas; i++)
        for(j=0; j<columnas; j++)
            M[i][j]=i+j;

    for(i=0; i<filas; i++)
        for(j=0; j<columnas; j++)
            s[i]+=v[i]*M[i][j];

    for(i=0; i<filas; i++)
        printf("s[%d]= %d ", i, s[i]);

    printf("\n");
}
```

CAPTURAS DE PANTALLA:



```
s[0]= 0s[1]= 15s[2]= 40s[3]= 75s[4]= 120adri@adri-virtual-machine ~/Descargas $
gcc -o pmv-secuencial pmv-secuencial.c
adri@adri-virtual-machine ~/Descargas $ ./pmv-secuencial 5 5
s[0]= 0 s[1]= 15 s[2]= 40 s[3]= 75 s[4]= 120
```

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).

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 i, j, filas, columnas, v[100], M[100][100], s[100];
    if(argc < 3) {
        fprintf(stderr, "Falta filas o columnas\n");
        exit(-1);
    }
    filas = atoi(argv[1]);
    columnas = atoi(argv[2]);
    #pragma omp parallel for
    for(i=0; i<filas; i++){
        v[i]=i;
        s[i]=0;
    }
    #pragma omp parallel for
    for(i=0; i<filas; i++)
        for(j=0; j<columnas; j++)
            M[i][j]=i+j;
    #pragma omp parallel for
    for(i=0; i<filas; i++)
        for(j=0; j<columnas; j++)
            s[i]+=v[i]*M[i][j];
    #pragma omp parallel for
    for(i=0; i<filas; i++)
        printf("s[%d]= %d ", i, s[i]);
    printf("\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 i, j, filas, columnas, v[100], M[100][100], s[100];

    if(argc < 3) {
        fprintf(stderr, "Falta filas o columnas\n");
        exit(-1);
    }

    filas = atoi(argv[1]);
    columnas = atoi(argv[2]);

    for(i=0; i<filas; i++){
        v[i]=i;
        s[i]=0;
    }

    for(i=0; i<filas; i++)
        #pragma omp parallel for
        for(j=0; j<columnas; j++)
            M[i][j]=i+j;

    for(i=0; i<filas; i++)
        #pragma omp parallel for reduction(+:s[i])
        for(j=0; j<columnas; j++)
            s[i]+=v[i]*M[i][j];

    #pragma omp parallel for
    for(i=0; i<filas; i++)
        printf("s[%d]= %d ", i, s[i]);

    printf("\n");
}

```

RESPUESTA:

Para el primero sólo hay que añadir `#pragma omp parallel for` antes del `for` de las filas, para el segundo igual pero además hay que añadir `reduction(+:s[i])` para el `for` de las columnas.

CAPTURAS DE PANTALLA:

```

adri@adri-virtual-machine ~/Descargas $ gcc -o pmv-OpenMP-a pmv-OpenMP-a.c
adri@adri-virtual-machine ~/Descargas $ ./pmv-OpenMP-a 5 5
s[0]= 0 s[1]= 15 s[2]= 40 s[3]= 75 s[4]= 120

adri@adri-virtual-machine ~/Descargas $ gcc -o pmv-OpenMP-b pmv-OpenMP-b.c
adri@adri-virtual-machine ~/Descargas $ ./pmv-OpenMP-b 5 5
s[0]= 0 s[1]= 15 s[2]= 40 s[3]= 75 s[4]= 120

```

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 <stdlib.h>
#include <stdio.h>
#include <omp.h>
#define PRINTF_ALL
int main(int argc, char **argv) {
    double resultado;
    unsigned int N=3, i, j;
    double cgt1, cgt2;
    double ncgt;
    if (argc<2) {
        printf("Faltan no componentes del vector\n");
        exit(-1);
    }
    N=atoi(argv[1]);
    double **matriz, *v1, *v2;
    v1=(double *) malloc(N*sizeof(double));
    v2=(double *) malloc(N*sizeof(double));
    matriz=(double **) malloc(N*sizeof(double *));
    for(i=0; i<N; i++) matriz[i]=(double *) malloc(N*sizeof(double));
    #pragma omp parallel
    {
        #pragma omp for
        for(i=0; i<N; i++) v1[i]=i;
        #pragma omp for
        for(i=0; i<N; i++) {
            for(j=0; j<N; j++) {
                matriz[i][j]=i+j;
            }
        }
    }
    cgt1=omp_get_wtime();
    for(i=0; i<N; i++) {
        resultado=0;
        #pragma omp parallel for reduction(+:resultado)
        for(j=0; j<N; j++) {
            resultado+=(matriz[i][j]*v1[j]);
        }
        v2[i]=resultado;
    }
    cgt2=omp_get_wtime();
    ncgt=cgt2-cgt1;
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n", ncgt, N);
    printf("Vector resultadoado\n");
    for(i=0; i<N; i++) printf("%f ", v2[i]);
    free(v1);
    free(v2);
    for(i=0; i<N; i++) free(matriz[i]);
    free(matriz);
}
```

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 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). Usar -O2 al compilar.

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

COMENTARIOS SOBRE LOS RESULTADOS: