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): Ernesto Serrano Collado
Grupo de prácticas: A2
Fecha de entrega: 16/04/2016
Fecha evaluación en clase: 22/04/2016

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 poner default(none) se debe especificar el ámbito de todas las variables involucradas en la región paralela, por lo que el compilador nos indica que hay que definir el ámbito de la variable n.

   CÓDIGO FUENTE: `shared-clauseModificado.c`

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

#ifdef _OPENMP
        #include <omp.h>
#else
        #define omp_get_thread_num() 0
        #define omp_get_num_threads() 1
#endif

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

```
gcc-4.9 -O2 -fopenmp -o bin/shared-clause src/shared-clause.c
src/shared-clause.c: In function 'main':
src/shared-clause.c:19:10: error: 'n' not specified in enclosing parallel
  #pragma omp parallel for shared(a) default(none)
          ^
src/shared-clause.c:19:10: error: enclosing parallel
make: *** [shared-clause] Error 1
bender:practica2 ernesto$
```

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 se inicializa fuera de parallel la variable contendrá basura (un valor indeteterminado), por lo que se debe inicializar dentro de la seccion paralela o usar firstprivate

CÓDIGO FUENTE: `private-clauseModificado.c`

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

#ifdef _OPENMP
        #include <omp.h>
#else
        #define omp_get_thread_num() 0
        #define omp_get_num_threads() 1
#endif

int main(int argc, char **argv)
{
        int i, n = 7;
        int a[n], suma;

        for (i=0; i<n; i++) a[i] = i;
        // suma=1; // incorrecto, si la inicializamos aquí luego contendra basura
        #pragma omp parallel private(suma)
        {
                suma=1; // Lugar correcto donde inicializar la variable
                #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:

variable suma inicializada a 1 fuera de la seccion paralela (incorrecto)

```
bender:practica2 ernesto$ bin/private-clauseModificado
thread 0 suma a[0]thread 1 suma a[2]thread 2 suma a[4]thread 3 suma a[6]thread 0 s
uma a[1]thread 1 suma a[3]thread 2 suma a[5]
* thread 3 suma= 1761617414
* thread 0 suma= 5
* thread 1 suma= 1761617413
* thread 2 suma= 1761617417
bender:practica2 ernesto$
```

variable suma inicializada a 1 dentro de la seccion paralela (correcto)

```
bender:practica2 ernesto$ bin/private-clauseModificado
thread 1 suma a[2]thread 2 suma a[4]thread 0 suma a[0]thread 3 suma a[6]thread 1 s
uma a[3]thread 2 suma a[5]thread 0 suma a[1]
* thread 1 suma= 6
* thread 2 suma= 10
* thread 3 suma= 7
* thread 0 suma= 2
bender:practica2 ernesto$
```

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

RESPUESTA: Que la variable suma pasará a ser compartida, con lo que los distintos hilos podrán modificarla y machacar el estado previo, este comportamiento sería incorrecto

CÓDIGO FUENTE: `private-clauseModificado3.c`

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

```
#ifdef _OPENMP
        #include <omp.h>
#else
        #define omp_get_thread_num() 0
        #define omp_get_num_threads() 1
#endif

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

```
bender:practica2 ernesto$ bin/private-clauseModificado3
thread 1 suma a[2]thread 2 suma a[4]thread 3 suma a[6]thread 0 suma a[0]thread 1 s
uma a[3]thread 2 suma a[5]thread 0 suma a[1]
* thread 3 suma= 15
* thread 1 suma= 15
* thread 2 suma= 15
* thread 0 suma= 15
bender:practica2 ernesto$ bin/private-clauseModificado3
thread 1 suma a[2]thread 3 suma a[6]thread 0 suma a[0]thread 2 suma a[4]thread 1 s
uma a[3]thread 0 suma a[1]thread 2 suma a[5]
* thread 1 suma= 13
* thread 0 suma= 13
* thread 3 suma= 13
* thread 2 suma= 13
```

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: Porque lastprivate asigna a la variable el ultimo valor que se asignaría en una ejecución secuencial, que siempre será 0 + 6

CAPTURAS DE PANTALLA:

```
Fuera de la construcción parallel suma=6
[bender:practica2 ernesto$ bin/firstlastprivate
 thread 2 suma a[4] suma=4
 thread 3 suma a[6] suma=6
 thread 0 suma a[0] suma=0
 thread 1 suma a[2] suma=2
 thread 2 suma a[5] suma=9
 thread 0 suma a[1] suma=1
 thread 1 suma a[3] suma=5

Fuera de la construcción parallel suma=6
[bender:practica2 ernesto$ bin/firstlastprivate
 thread 2 suma a[4] suma=4
 thread 3 suma a[6] suma=6
 thread 0 suma a[0] suma=0
 thread 1 suma a[2] suma=2
 thread 2 suma a[5] suma=9
 thread 0 suma a[1] suma=1
 thread 1 suma a[3] suma=5

Fuera de la construcción parallel suma=6
bender:practica2 ernesto$
```

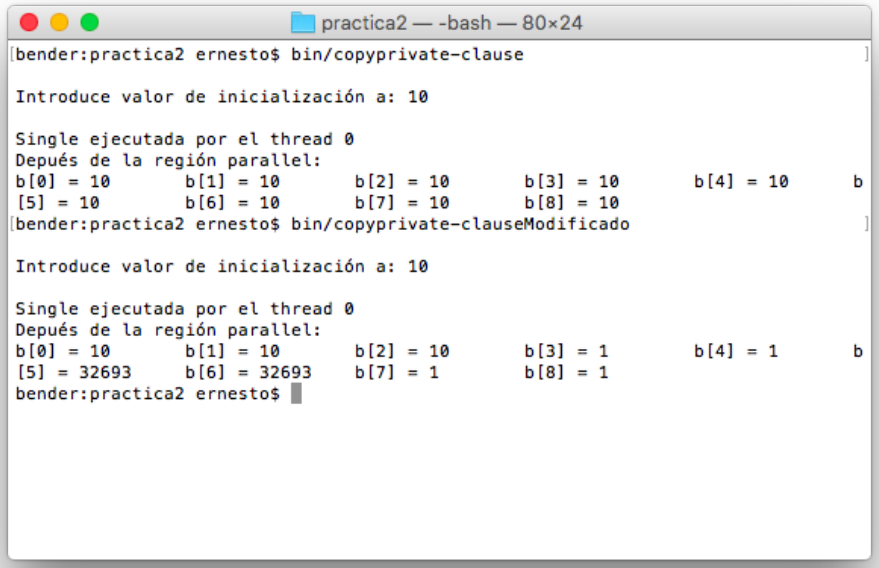
5. ¿Qué ocurre si en `copyprivate-clause.c` se elimina la cláusula `copyprivate(a)` en la directiva `single`? ¿A qué cree que es debido?

RESPUESTA: Que el funcionamiento es incorrecto, estamos usando la variable "a" que en algunos casos contiene basura porque la hemos leido en un thead pero no la hemos copiado a los demas thread, para esto sirve copyprivate, una vez termina el single, copia la variable a los demas threads mediante (difusión)

CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

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

#ifdef _OPENMP
        #include <omp.h>
#else
        #define omp_get_thread_num() 0
        #define omp_get_num_threads() 1
#endif

int main(int argc, char **argv)
{
        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("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:



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

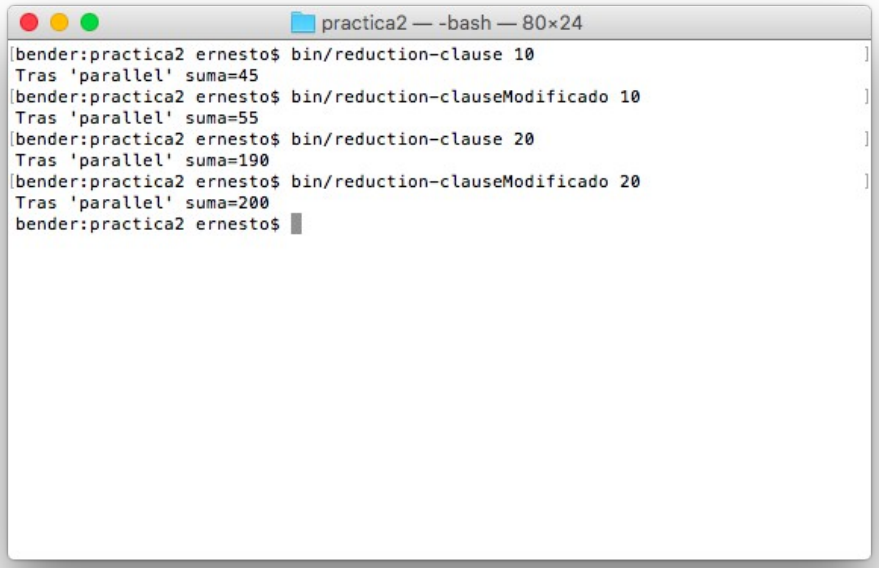
RESPUESTA: El programa suma todos los números desde 0 hasta n (maximo 20) y los almacena en la variable suma, si hacemos el cambio la variable tendrá un valor inicial de 10, por lo que la suma final será el valor mas 10.

CÓDIGO FUENTE: `reduction-clauseModificado.c`

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

#ifdef _OPENMP
	#include <omp.h>
#else
	#define omp_get_thread_num() 0
	#define omp_get_num_threads() 1
#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:



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 sin usar directivas de trabajo compartido .

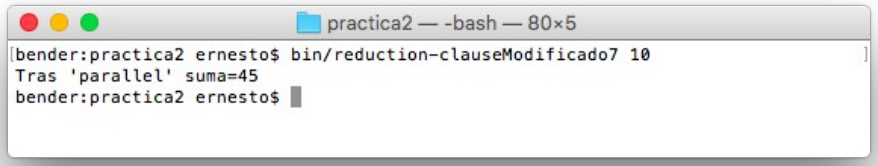
RESPUESTA: Si no usamos directivas de trabajo compartido debemos tener alguna forma de separar el código para que cada thread haga una parte (al indicar paralle ese trozo se ejecutará en todas las hebras), se ha optado por jugar con el identificador de hebra y el numero total de hebras y se ha usado un round-robin siguiendo la sugerencia del profesor. Es muy importante tener en cuenta que la variable i tiene que ser privada para cada hebra.

CÓDIGO FUENTE: `reduction-clauseModificado7.c`

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

#ifdef _OPENMP
        #include <omp.h>
#else
        #define omp_get_thread_num() 0
        #define omp_get_num_threads() 1
#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 private(i) reduction(+:suma)
        {
                for (i=omp_get_thread_num(); i<n; i+=omp_get_num_threads())
                        suma += a[i];

        }

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

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):

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

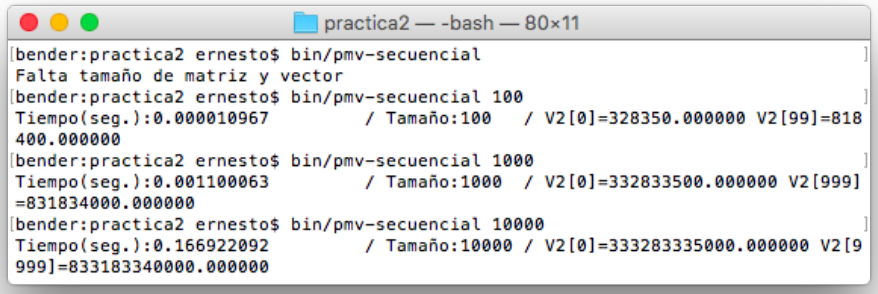
NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada al programa; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños 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` **`NOTA:`** `la version con variables globales está en pmv-secuencialGlobales.c`

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>

#ifdef _OPENMP
        #include <omp.h>
#else
        #define omp_get_thread_num() 0
        #define omp_get_num_threads() 1
#endif

int main(int argc, char** argv)
{
        int i, j;
        double t1, t2, total;
```

```
        //Leer argumento de entrada (no de componentes del vector)
        if (argc<2){
                printf("Falta tamaño de matriz y vector\n");
                exit(-1);
        }

        unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned
int) = 4 B)

        double *v1, *v2, **M;
        v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
        v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc
devuelve NULL
        M = (double**) malloc(N*sizeof(double *));
        if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
                printf("Error en la reserva de espacio para los vectores\n");
                exit(-2);
        }

        for (i=0; i<N; i++){
                M[i] = (double*) malloc(N*sizeof(double));
                if ( M[i]==NULL ){
                        printf("Error en la reserva de espacio para los vectores\n");
                        exit(-2);
                }
        }
        //A partir de aqui se pueden acceder las componentes de la matriz como M[i][j]

        //Inicializar matriz y vectores
        for (i=0; i<N;i++)
        {
                v1[i] = i;
                v2[i] = 0;
                for(j=0;j<N;j++)
                        M[i][j] = i+j;
        }

        //Medida de tiempo
        t1 = omp_get_wtime();

        //Calcular producto de matriz por vector v2 = M · v1
        for (i=0; i<N;i++)
                for(j=0;j<N;j++)
                                v2[i] += M[i][j] * v1[j];

        //Medida de tiempo
        t2 = omp_get_wtime();
        total = t2 - t1;

        //Imprimir el resultado y el tiempo de ejecución
        printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n",
total,N,v2[0],N-1,v2[N-1]);

        // Imprimir todos los componentes de v2 (solo si es razonable el tamaño)
        if (N<20)
                for (i=0; i<N;i++)
                        printf(" V2[%d]=%5.2f\n", i, v2[i]);

        free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2
        for (i=0; i<N; i++)
                free(M[i]);
        free(M);

        return 0;
}
```

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).

CÓDIGO FUENTE : `pmv-OpenMP-a.c`

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>

#ifdef _OPENMP
        #include <omp.h>
#else
        #define omp_get_thread_num() 0
        #define omp_get_num_threads() 1
#endif

int main(int argc, char** argv)
{
        int i, j;
        double t1, t2, total;

        //Leer argumento de entrada (no de componentes del vector)
        if (argc<2){
                printf("Falta tamaño de matriz y vector\n");
                exit(-1);
        }
```

```
        unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned
int) = 4 B)

        double *v1, *v2, **M;
        v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
        v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc
devuelve NULL
        M = (double**) malloc(N*sizeof(double *));
        if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
                printf("Error en la reserva de espacio para los vectores\n");
                exit(-2);
        }

        for (i=0; i<N; i++){
                M[i] = (double*) malloc(N*sizeof(double));
                if ( M[i]==NULL ){
                        printf("Error en la reserva de espacio para los vectores\n");
                        exit(-2);
                }
        }
        //A partir de aqui se pueden acceder las componentes de la matriz como M[i][j]

        //Inicializar matriz y vectores
        #pragma omp parallel
        {
                #pragma omp for private(j)
                for (i=0; i<N;i++)
                {
                        v1[i] = i;
                        v2[i] = 0;
                        for(j=0;j<N;j++)
                                M[i][j] = i+j;
                }

                //Medida de tiempo
                #pragma omp single
                t1 = omp_get_wtime();

                //Calcular producto de matriz por vector v2 = M · v1

                #pragma omp for private(j)
                for (i=0; i<N;i++)
                        for(j=0;j<N;j++)
                                v2[i] += M[i][j] * v1[j];

                //Medida de tiempo
                #pragma omp single
                t2 = omp_get_wtime();
        }

        total = t2 - t1;

        //Imprimir el resultado y el tiempo de ejecución
        printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n",
total,N,v2[0],N-1,v2[N-1]);

        // Imprimir todos los componentes de v2 (solo si es razonable el tamaño)
        if (N<20)
                for (i=0; i<N;i++)
                        printf(" V2[%d]=%5.2f\n", i, v2[i]);



        free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2
        for (i=0; i<N; i++)
                free(M[i]);
        free(M);

        return 0;
}
```

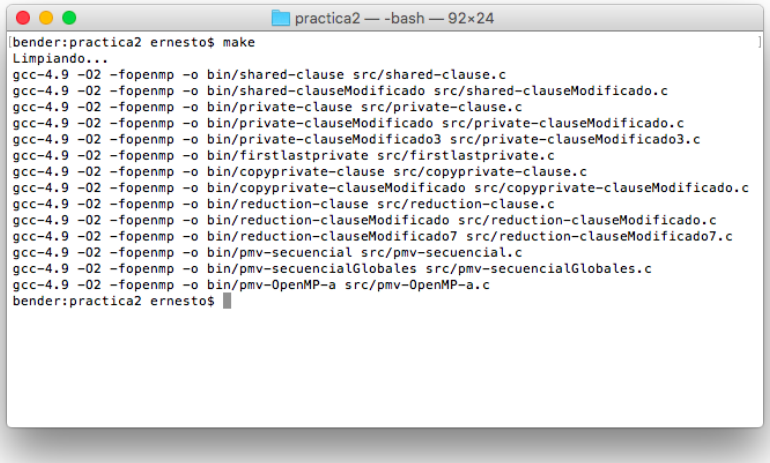
CÓDIGO FUENTE: `pmv-OpenMP-b.c`

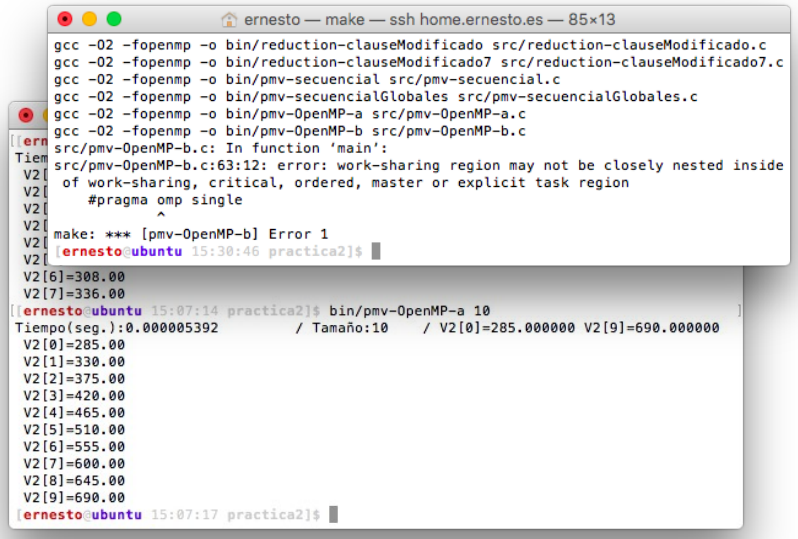
```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>

#ifdef _OPENMP
	#include <omp.h>
#else
	#define omp_get_thread_num() 0
	#define omp_get_num_threads() 1
#endif

int main(int argc, char** argv)
{
	int i, j;
	double t1, t2, total;

	//Leer argumento de entrada (no de componentes del vector)
	if (argc<2){
		printf("Falta tamaño de matriz y vector\n");
		exit(-1);
	}

	unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned
int) = 4 B)

	double *v1, *v2, **M;
	v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
	v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc
devuelve NULL
	M = (double**) malloc(N*sizeof(double *));
	if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
		printf("Error en la reserva de espacio para los vectores\n");
		exit(-2);
	}

	for (i=0; i<N; i++){
		M[i] = (double*) malloc(N*sizeof(double));
		if ( M[i]==NULL ){
			printf("Error en la reserva de espacio para los vectores\n");
			exit(-2);
		}
	}
	//A partir de aqui se pueden acceder las componentes de la matriz como M[i][j]

	//Inicializar matriz y vectores

	for (i=0; i<N;i++)
	{
		v1[i] = i;
		v2[i] = 0;
		#pragma omp parallel for
		for(j=0;j<N;j++)
			M[i][j] = i+j;
	}

	//Medida de tiempo
	t1 = omp_get_wtime();

	//Calcular producto de matriz por vector v2 = M · v1
	for (i=0; i<N;i++)
	{
		#pragma omp parallel
		{
			double tmp = 0; // aqui nos guardamos el valor temporalmente
			#pragma omp for
			for(j=0;j<N;j++)
			{
				//EL PROFESOR NO QUIERE QUE USEMOS ATOMIC AQUI
				tmp += M[i][j] * v1[j];
			}
```

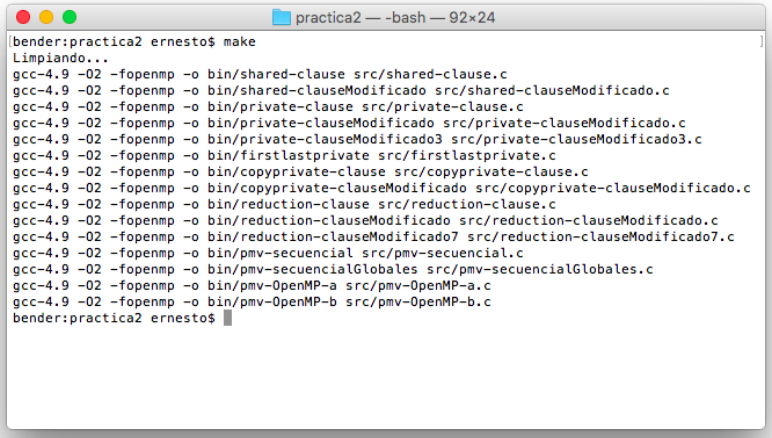
```
                        #pragma omp critical
                        v2[i] += tmp;
                }
        }

        //Medida de tiempo
        t2 = omp_get_wtime();

        total = t2 - t1;

        //Imprimir el resultado y el tiempo de ejecución
        printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n",
total,N,v2[0],N-1,v2[N-1]);

        // Imprimir todos los componentes de v2 (solo si es razonable el tamaño)
        if (N<20)
                for (i=0; i<N;i++)
                        printf(" V2[%d]=%5.2f\n", i, v2[i]);


        free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2
        for (i=0; i<N; i++)
                free(M[i]);
        free(M);

      return 0;
}
```

RESPUESTA: No he tenido errores de compilación, en la versión A todo ha funcionado sin problemas, dando los mismos resultados que la versión secuencial pero haciendo uso de paralelismo, la versión B si me ha devuelto resultados erróneos que se han solucionado haciendo que fuera atómica la escritura en v2, pero el profesor indicó que eso no es eficiente así que lo he intentado solucionar poniendo un single pero evidentemente no puedo poner un omp single dentro de un omp for y me daba un error de compilación, al final he optado por usar critical, que para esta operación es mucho mas eficiente que atomic.

CAPTURAS DE PANTALLA:




ernesto — make — ssh home.ernesto.es — 85×13
gcc -O2 -fopenmp -o bin/reduction-clauseModificado src/reduction-clauseModificado.c
gcc -O2 -fopenmp -o bin/reduction-clauseModificado7 src/reduction-clauseModificado7.c
gcc -O2 -fopenmp -o bin/pmv-secuencial src/pmv-secuencial.c
gcc -O2 -fopenmp -o bin/pmv-secuencialGlobales src/pmv-secuencialGlobales.c
gcc -O2 -fopenmp -o bin/pmv-OpenMP-a src/pmv-OpenMP-a.c
gcc -O2 -fopenmp -o bin/pmv-OpenMP-b src/pmv-OpenMP-b.c
src/pmv-OpenMP-b.c: In function 'main':
src/pmv-OpenMP-b.c:63:12: error: work-sharing region may not be closely nested inside
of work-sharing, critical, ordered, master or explicit task region
#pragma omp single
^
make: *** [pmv-OpenMP-b] Error 1
[ernesto@ubuntu 15:30:46 practica2]$
V2[6]=308.00
V2[7]=336.00
[ernesto@ubuntu 15:07:14 practica2]$ bin/pmv-OpenMP-a 10
Tiempo(seg.):0.000005392 / Tamaño:10 / V2[0]=285.000000 V2[9]=690.000000
V2[0]=285.00
V2[1]=330.00
V2[2]=375.00
V2[3]=420.00
V2[4]=465.00
V2[5]=510.00
V2[6]=555.00
V2[7]=600.00
V2[8]=645.00
V2[9]=690.00
[ernesto@ubuntu 15:07:17 practica2]$


practica2 — -bash — 92×24
[bender:practica2 ernesto$ make
Limpiando...
gcc-4.9 -O2 -fopenmp -o bin/shared-clause src/shared-clause.c
gcc-4.9 -O2 -fopenmp -o bin/shared-clauseModificado src/shared-clauseModificado.c
gcc-4.9 -O2 -fopenmp -o bin/private-clause src/private-clause.c
gcc-4.9 -O2 -fopenmp -o bin/private-clauseModificado src/private-clauseModificado.c
gcc-4.9 -O2 -fopenmp -o bin/private-clauseModificado3 src/private-clauseModificado3.c
gcc-4.9 -O2 -fopenmp -o bin/firstlastprivate src/firstlastprivate.c
gcc-4.9 -O2 -fopenmp -o bin/copyprivate-clause src/copyprivate-clause.c
gcc-4.9 -O2 -fopenmp -o bin/copyprivate-clauseModificado src/copyprivate-clauseModificado.c
gcc-4.9 -O2 -fopenmp -o bin/reduction-clause src/reduction-clause.c
gcc-4.9 -O2 -fopenmp -o bin/reduction-clauseModificado src/reduction-clauseModificado.c
gcc-4.9 -O2 -fopenmp -o bin/reduction-clauseModificado7 src/reduction-clauseModificado7.c
gcc-4.9 -O2 -fopenmp -o bin/pmv-secuencial src/pmv-secuencial.c
gcc-4.9 -O2 -fopenmp -o bin/pmv-secuencialGlobales src/pmv-secuencialGlobales.c
gcc-4.9 -O2 -fopenmp -o bin/pmv-OpenMP-a src/pmv-OpenMP-a.c
gcc-4.9 -O2 -fopenmp -o bin/pmv-OpenMP-b src/pmv-OpenMP-b.c
bender:practica2 ernesto$


ernesto — bin/pmv-OpenMP-b 10 — ssh home.ernesto.es — 85×13
[ernesto@ubuntu 15:21:51 practica2]$ bin/pmv-OpenMP-b 10
Tiempo(seg.):0.000024507 / Tamaño:10 / V2[0]=285.000000 V2[9]=690.000000
V2[0]=285.00
V2[1]=330.00
V2[2]=375.00
V2[3]=420.00
V2[4]=465.00
V2[5]=510.00
V2[6]=555.00
V2[7]=600.00
V2[8]=645.00
V2[9]=690.00
[ernesto@ubuntu 15:21:53 practica2]$

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`

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>

#ifdef _OPENMP
	#include <omp.h>
#else
	#define omp_get_thread_num() 0
	#define omp_get_num_threads() 1
#endif

int main(int argc, char** argv)
{
	int i, j;
	double t1, t2, total;

	//Leer argumento de entrada (no de componentes del vector)
	if (argc<2){
		printf("Falta tamaño de matriz y vector\n");
		exit(-1);
	}

	unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned
int) = 4 B)

	double *v1, *v2, **M;
	v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
	v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc
devuelve NULL
	M = (double**) malloc(N*sizeof(double *));
	if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
		printf("Error en la reserva de espacio para los vectores\n");
		exit(-2);
	}

	for (i=0; i<N; i++){
		M[i] = (double*) malloc(N*sizeof(double));
		if ( M[i]==NULL ){
			printf("Error en la reserva de espacio para los vectores\n");
			exit(-2);
		}
	}
	//A partir de aqui se pueden acceder las componentes de la matriz como M[i][j]

	//Inicializar matriz y vectores
	for (i=0; i<N;i++)
	{
		v1[i] = i;
		v2[i] = 0;
		#pragma omp parallel for
		for(j=0;j<N;j++)
			M[i][j] = i+j;
	}

	//Medida de tiempo
	t1 = omp_get_wtime();
```

```
        //Calcular producto de matriz por vector v2 = M · v1


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

        //Medida de tiempo
        t2 = omp_get_wtime();
        total = t2 - t1;

        //Imprimir el resultado y el tiempo de ejecución
        printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n",
total,N,v2[0],N-1,v2[N-1]);

        // Imprimir todos los componentes de v2 (solo si es razonable el tamaño)
        if (N<20)
                for (i=0; i<N;i++)
                        printf(" V2[%d]=%5.2f\n", i, v2[i]);


        free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2
        for (i=0; i<N; i++)
                free(M[i]);
        free(M);

        return 0;
}
```

RESPUESTA: La versión con reduction simplifica mucho el código al no tener que sumar en una variable, aunque eso si, nos hemos tenido que almacenar la suman en una variable porque OpenMP no permite hacer reduction en un elemento de un vector, así que hacemos reduction sobre una variable y luego asignamos esa variable al elemento del vector.

CAPTURAS DE PANTALLA:

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 local 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.

    TABLA Y GRÁFICA (por *ejemplo* para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: algúno del orden de cientos de miles):

Pc local

Tamaño	Filas	Columnas	Reduction
100	0,002048277	0,00043671	0,000372993
500	0,002619774	0,002407056	0,004969974
1000	0,006333042	0,00531748	0,009130735
5000	0,074021248	0,082197199	0,198913174
10000	0,344622625	0,319465929	0,797889673
50000			

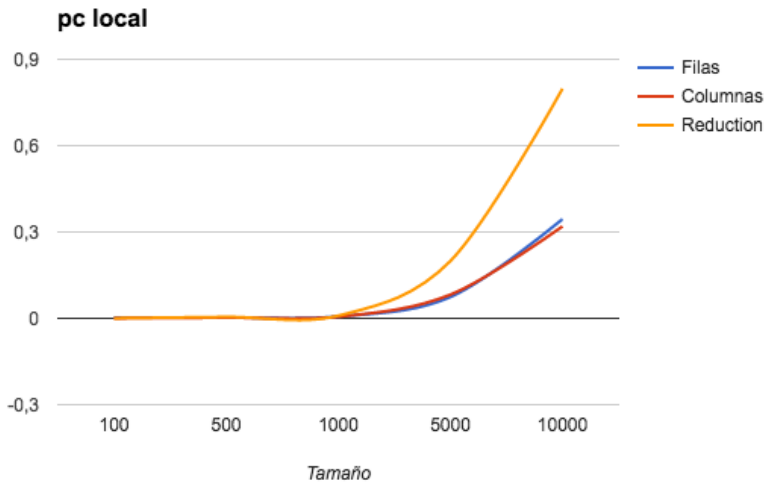
mac

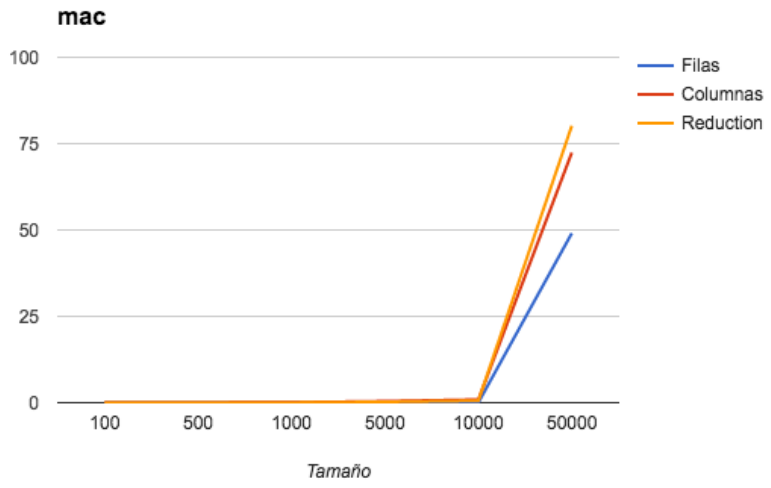
Tamaño	Filas	Columnas	Reduction
100	0,000071049	0,007724047	0,003407955
500	0,00027585	0,039107084	0,018273115
1000	0,003185987	0,084686995	0,037081003
5000	0,021971226	0,412250996	0,224373102
10000	0,044157028	0,889763117	0,563114166
50000	49,06640315	72,40687203	80,187814

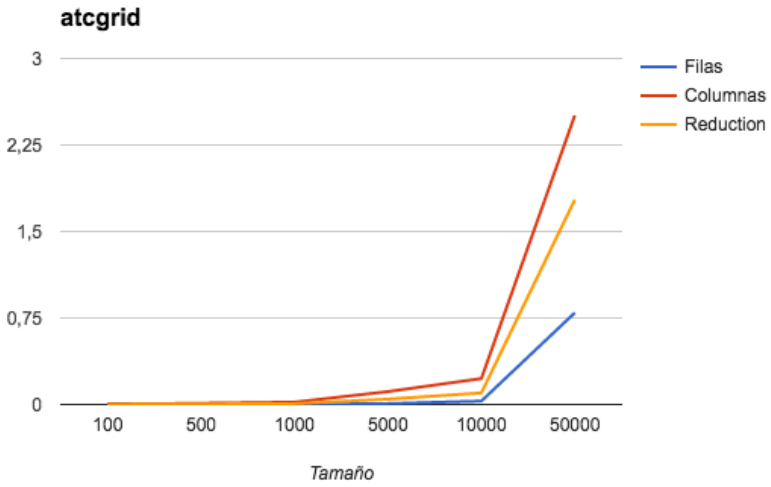
atcgrid

Tamaño	Filas	Columnas	Reduction
100	0,005737335	0,002009444	0,000775968
500	0,001909175	0,010613414	0,004007339

1000	0,000146119	0,020439962	0,007616263
5000	0,008247251	0,111904285	0,046526796
10000	0,029457262	0,224905885	0,10056217
50000	0,795177555	2,506143905	1,775972514







COMENTARIOS SOBRE LOS RESULTADOS:

Para la realización de este ejercicio se ha contado con una serie de scripts que también se incluyen en el cuaderno de prácticas.

No he podido realizar la prueba con 100.000 componentes ni en mi pc-local ni en atcgrid ya que en mi pc tardaba una barbaridad y en atcgrid me tiraba fuera al pasarme del tiempo máximo. Lo que si he podido ejecutar es 50000 en atcgrid, pero no en mi pc-local.

Tambien he ejecutado las pruebas en un Mac con un core i7, ahi si terminaban las 50000 pero aun así ni punto de comparación con el grid

He podido apreciar que por filas es mucho mas eficiente, si lo hacemos por columnas, usar reduction es menos eficiente que montarlo a mano con critial, supongo que por la parafernalia que tiene que montar el compilador para gestionarlo, eso si, es mucho mas eficiente que usar atomic.

Se nota que para tareas intensivas y con gran numero de iteraciones el nodo de supercomputación es muchísimo mas eficiente con diferencia.