Seminario 2. Programación con OpenMP

1. Completa el siguiente programa en C para que cada hilo muestre por pantalla un mensaje con su identificador:

```
#include <stdio.h>
#include …
int main() {
    …
    printf("Soy el hilo %d\n", …);
    return 0;
}
```

2. Paralelizar la siguiente función mediante OpenMP que, de acuerdo con la siguiente expresión, obtiene el vector *z* como combinación lineal de dos vectores *x* e *y* de *n* componentes y dos números reales *a* y *b*:

$$z = a * x + b * y .$$

```
void combinacion_lineal(double x[], double y[], double z[], double a, double b, int n) {
    int i;
    …
    for (i=0; i<n; i++)
        z[i] = a*x[i] + b*y[i];
}
```

3. Paraleliza de nuevo la función del ejercicio anterior teniendo en cuenta la modificación realizada en el código que a continuación se muestra:

```
void combinacion_lineal(double x[], double y[], double z[], double a, double b, int n) {
    int i;
    double temp;
    …
    for (i=0; i<n; i++) {
        temp=a*x[i]+b*y[i];
        z[i]=temp;
    }
}
```

4. Incorpora al siguiente programa en C todas las instrucciones necesarias para que el hilo 0 muestre por pantalla el número total de hilos que participan en su ejecución.

```
#include <stdio.h>
#include …
int main() {
    …
    printf("Somos %d hilos\n", ….);
```

```
    …
    return 0;
}
```

5. Modifica el programa anterior para que el código sea compilado satisfactoriamente por un compilador que interprete o no las directivas de OpenMP.

6. La siguiente función calcule el producto escalar de dos vectores. Paralelízala mediante OpenMP:

```
double producto_escalar(double x[ ], double y[ ], int n) {
    int i;
    double suma;
    suma=0;
    …
    for (i=0; i<n; i++)
        suma = suma + x[i]*y[i];
    return suma;
}
```

7. Paraleliza la siguiente función que devuelve el factorial de un número que se proporciona como dato de entrada:

```
double factorial(int n) {
    int i;
    double f;
    /* Cálculo del factorial */
    f=1;
    …
    for (i=2; i<=n; i++)
      f*=i;
    return f;
}
```

8. Supongamos que ejecutamos código que a continuación se muestra con dos hilos. ¿Qué mensajes se mostrarían por pantalla?

```
int main () {
    int a=5;
    #pragma omp parallel private (a)
    {
        a=8;
        printf("Soy hilo %d. Valor de a = %d\n",omp_get_thread_num(),a);
    }
```

```
    printf("Soy hilo %d. Valor de a = %d\n",omp_get_thread_num(),a);
    #pragma omp parallel private (a)
    {
        printf("Soy hilo %d. Valor de a = %d\n",omp_get_thread_num(),a);
    }
    #pragma omp parallel
    {
        printf("Soy hilo %d. Valor de a = %d\n",omp_get_thread_num(),a);
    }
}
```

Solución

Soy hilo …. Valor de a = …
Soy hilo …. Valor de a = …

Soy hilo …. Valor de a = …

Soy hilo …. Valor de a = …
Soy hilo …. Valor de a = …

Soy hilo ….. Valor de a = …
Soy hilo …. Valor de a = …

9. Dado el código de la siguiente función encargada de calcular el producto de una matriz por un vector y obtener la suma de los elementos del vector resultante:

```
double funcion(double A[M][N], double x[N], double y[M]) {
    int i, j;
    double suma, suma2;
    suma2 = 0;
    for (i=0; i<M; i++) {
        suma = 0;
        for (j=0; j<N; j++) {
            suma = suma + A[i][j]*x[j];
        }
        y[i] = suma;
        suma2 = suma2 + y[i];
    }
    return suma2;
}
```

a) Realiza una implementación paralela en la que se repartan las iteraciones del bucle externo.

```
double funcion(double A[M][N], double x[N], double y[M]) {
```

```
    int i, j;
    double suma, suma2;
    suma2 = 0;
    …
    for (i=0; i<M; i++) {
        suma = 0;
        for (j=0; j<N; j++) {
            suma = suma + A[i][j]*x[j];
        }
        y[i] = suma;
        suma2 = suma2 + y[i];
    }
    return suma2;
}
```

b) Realiza una implementación paralela en la que se repartan las iteraciones del bucle interno.

```
double funcion(double A[M][N], double x[N], double y[M]) {
    int i, j;
    double suma, suma2;
    suma2 = 0;
    for (i=0; i<M; i++) {
        suma = 0;
        …
        for (j=0; j<N; j++) {
            suma = suma + A[i][j]*x[j];
        }
        y[i] = suma;
        suma2 = suma2 + y[i];
    }
    return suma2;
}
```

10. Paraleliza eficientemente la siguiente función mediante OpenMP:

```
double funcion(double A[M][N]) {
    int i, j;
    double prod;
    …
    for (i=1; i<M; i++) {
        for (j=0; j<N; j++) {
            A[i][j] = 2.0 * A[i-1][j];
        }
```

```
    }
    prod = 1.0;
    …
    for (i=0; i<M; i++) {
        for (j=0; j<N; j++) {
            prod = prod * A[i][j];
        }
    }
    return prod;
}
```

11. En un programa llamado “mi_programa” y paralelizado mediante OpenMP, aparece la siguiente directiva:

```
#pragma omp parallel for
```

Incorpora las cláusulas necesarias para que el reparto de iteraciones:

a) Sea estático con bloques de igual longitud.

```
#pragma omp parallel for …
```

b) Sea estático y cíclico.

```
#pragma omp parallel for …
```

c) Sea estático y cíclico con bloques de longitud igual a 10.

```
#pragma omp parallel for …
```

d) Sea cíclico y dinámico con bloques de longitud igual a 5.

```
#pragma omp parallel for …
```

e) Se defina en tiempo de ejecución y sea estático y cíclico con bloques de longitud igual a 2.

```
#pragma omp parallel for …
```

12. Paralelizar mediante OpenMP el siguiente fragmento de código. Cada hilo deberá mostrar por pantalla su identificador y el número total de iteraciones del bucle que le han sido asignadas.

```
int main() {
    …
    for (i=0; i<n; i++) {
        a=funcion(i*4);
        v[i]=a+2;
    }
     ...
}
```

13. Paraleliza la función del ejercicio 6 que calculaba el producto escalar de 2 vectores:

a) Usando la directiva for sin reducción.
b) Sin usar la directiva for.

14. Calcular el coste de los siguientes fragmentos de código:

a)

```
…
for (i=1; i<=n; i++)
    x = x +2*y;
for (j=1; j<=n; j++)
    z = z + x;
…
```

b)

```
…
for (i=1; i<=n; i++) {
    x = x +2*y;
    for (j=1; j<=n; j++)
        z = z + x;
}
…
```