Practico 7

Alumno : Federico A. Verges.

Aquí estan los ejercicios para analizar.

Ejercicio 2 Comparacion de algoritmos OpenMPI

Diferencias

¿Qué diferencia hay en la ejecución de estos dos programas?

- En la Ejecución del primer algoritmo se ejecuta el for en todos los procesos 10 veces, es decir los 5 procesos ejecutan el ciclo "for" 10 veces (40 carteles por pantalla en total).
- En la ejecucion del segundo algoritmo el ciclo "for" se divide en todos los threads creados por lo tanto hay 10 carteles por pantalla divididos en los 5 threads creados.

¿Qué sucedería si la variable n no fuera privada?

- En el caso del primero programa utlizando la clausula "shared" donde la variable n deja de ser privada, el loop for se ejecuta 14 veces, es decir que un proceso realiza las 10 iteraciones y luego cada proceso hace una iteracion. Esto se debe a que la variable n se encuentra en una region paralela de memoria que es comun para todos los procesos.
- En el 2do algotimo funciona exactamente igual, ya que al tener la variable compartida, divide de igual manera la iteración.

Nota: Preguntar sobre como funciona el caso de una variable shared en un for.

Ejercicio 3 Suceptibilidad de paralelización

Código 1

```
DO i=1,N
   a[i]= a[i+1] + x
END DO
```

En este caso este codigo, el bucle es totalmente independiente por lotanto no sufre suceptiblidad al ser paralelzado.

Código 2

```
DO i=1,N
   a[i]= a[i] + b[i]
END DO
```

En este caso este codigo, el bucle es totalmente independiente por lotanto no sufre suceptiblidad al ser paralelzado.

Código 3

```
ix = base
DO i=1,N
   a ( ix ) = a ( ix ) - b ( i )
   ix = ix + stride
END DO
```

En este caso, en el bulcle no es independeinte debido a la base "ix", por lo tanto, al ser paralelizado, ambos threads acederan a la misma base por lo tanto modifican el mismo valor de la estructura "a".

Código 4

```
DO i=1, n
   b ( i ) = ( a ( i ) - a (i-1) )*0.5
END DO
```

En este caso, el bucle sobre la estructura "a" es independiente, por lo tanto es paralelizable, el unico inconveniente es que si a(0) no tiene nada, el compilador daria como resultado un error.

Ejercicio 4 Bloques estructurados.

Codigo 1

```
for (i=0;i<n;i++) {
    a[i] = 2.3*i;
    if (a[i] < b[i]) break;
}
```

En el caso del uso del break éste fuerza a que el thread finalice, por lo tanto si corta la ejecución de los otros threads para que todos terminen al mismo tiempo puede causar problemas ya que no se sabe que datos estaban modificando los demas threads, por lo tanto esto podria causar problemas en el codigo, esto provoca que sea suceptible al paralelismo. Cabe destacar que OpenMP no deja utilizar la sentencia break en los "for loop" provistos por OpenMP.

Codigo 2

```
flag = 0;
for (i=0;(i<n) & (!flag);i++){
    a[i] = 2.3*i;
```

```
    if (a[i] < b[i]) flag = 1;
}
```

En este caso, el thread que ingrese al if, modificará el valor del flag logrando que finalice la ejecucion del ciclo for.

Ejercicio 7 Planificación de Bucles

Programa 1

En este programa podemos ver la sentencia **"Schedule"** que sirve para definir estategias de reparto de tareas, en esta ocasión utiliza la sentencia **"static"** para indicar que divide el ciclo for en partes de k, y son repartidas por cada hilo, en este caso lo hace con 4, entonces el programa asigna una de esas partes de 4 por cada hilo que el programa tiene disponible y cuando lo hace en el ultimo hilo vuelve a empezar hasta que finaliza con el ciclo for.

Codigo:

```
#include <omp.h>
#include <stdio.h>
#include <unistd.h>
#define N 40
int main()
{
    int tid;
    int A[N];
    int i;
    for (i = 0; i < N; i++)
        A[i] = -1;
#pragma omp parallel for schedule(static, 4) private(tid)
    for (i = 0; i < N; i++)
    {
        tid = omp_get_thread_num();
        A[i] = tid;
        usleep(1);
    }
    for (i = 0; i < N / 2; i++)
        printf(" %2d", i);
    printf("\n");
    for (i = 0; i < N / 2; i++)
        printf(" %2d", A[i]);
    printf("\n\n\n");
    for (i = N / 2; i < N; i++)
        printf(" %2d", i);
    printf("\n");
    for (i = N / 2; i < N; i++)
        printf(" %2d", A[i]);
    printf("\n\n");
}
```

Salida:

```
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
  0  0  0  0  1  1  1  1  2  2  2  2  3  3  3  3  4  4  4  4


  20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
  5  5  5  5  6  6  6  6  7  7  7  7  0  0  0  0  1  1  1  1
```

Programa 2

En este caso el programa divide la iteracion for en partes de k igual que el programa anterior, pero en este caso las asigna dinamicamente a los hilos que tenemos disponibles, por lo tanto **El orden de la asignacion dependerá de la ejecución del programa**.

Codigo:

```
#include <omp.h>
#include <stdio.h>
#include <unistd.h>
#define N 40
int main()
{
    int tid;
    int A[N];
    int i;
    for (i = 0; i < N; i++)
        A[i] = -1;
#pragma omp parallel for schedule(dynamic, 4) private(tid)
    for (i = 0; i < N; i++)
    {
        tid = omp_get_thread_num();
        A[i] = tid;
        usleep(1);
    }
    for (i = 0; i < N / 2; i++)
        printf(" %2d", i);
    printf("\n");
    for (i = 0; i < N / 2; i++)
        printf(" %2d", A[i]);
    printf("\n\n\n");
    for (i = N / 2; i < N; i++)
        printf(" %2d", i);
    printf("\n");
    for (i = N / 2; i < N; i++)
        printf(" %2d", A[i]);
    printf("\n\n");

}
```

Salida:

```
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 0  0  0  0  4  4  4  4  3  3  3  3  7  7  7  7  6  6  6  6


20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
 1  1  1  1  2  2  2  2  7  7  7  7  2  2  2  2  0  0  0  0
```