ARQUITECTURA DE SISTEMAS PARALELOS
Muliprocesadores de memoria

Problemas Tema 3: Paralelización de bucles

3.1.- El siguiente bucle se va a ejecutar en una máquina multiprocesador:

```
do i = 1, N-2
        (1) A(i) =B(i)
        (2) C(i) = A(i) + B(i-1)
        (3) D(i) = C(i+1)
        (4) B(i) = C(i) * 2
enddo
```

Genere el grafo de dependencias
Escribe una versión paralela de dicho bucle y suponiendo que el tiempo de ejecución de una instrucción es T, ¿Cuántas veces más rápido es la ejecución al paralelizar en P procesadores?.¿Cuál será la máxima aceleración con infinitos procesadores y despreciando el tiempo de sincronización.

3.2. Represente el grafo de dependencias de los siguientes bucles:

a)

```
do i = 1, N-2
        (1) A(i) =B(i) +2
        (2) C(i) = A(i-2) +A(i+1)
enddo
```

b)

```
do i = 1, N-2
do j = 1, N-2
        (1) A(i,j) =A(i,j-1) *2
        (2) C(i,j) = A(i-2.j+1) +1
enddo
enddo
```

3.3.- El siguiente bucle se va a ejecutar en una máquina SMP de 4 procesadores:

```
do i = 1, 994
        (1) A(i) = D(i) - 10
        (2) B(i+1) = B(i+1) * A(i-1)
        (3) D(i+1) = D(i+1) + D(i+1)
Enddo
```

a) Genere el grafo de dependencias y el espacio de iteraciones correspondientes al bucle. Escribe una versión paralela de dicho bucle que pueda ejecutarse en P procesadores sin tener que utilizar funciones de sincronización.
b) Para el caso de esta máquina de 4 procesadores, escribe el código que ejecutará cada procesador (el bucle paralelo) si el reparto de las iteraciones entre los procesadores es (1) estático consecutivo; y (2) dinámico por chunks (trozos), con trozos de 50 iteraciones. ¿Cuántas iteraciones ejecutará el procesador P1 en el primer caso? ¿y en el segundo?

3.4- Para cada uno de los siguientes bucles indique en qué casos existen dependencia entre las dos instrucciones dadas. Justifique la respuesta.

a)

```
do i = 1, 100
      A(2*i) = ...
      ... = A(2*i+1)
enddo
```

b)

```
do i = 5, 100
      A(i-5) = ...
      ... = A(2*i+90
enddo
```

c)

```
do i = 1, 100
      A(3*i+100) = ...
      ... = A(2*i-1)
enddo
```

d)

```
do i = 1, 100
      A(6*i+3) = ...
      ... = A(3*i+81)
enddo
```

3.5.- Indique cómo paralelizar de la manera más efectiva el bucle siguiente para una máquina con 40 procesadores. Escriba el pseudocódigo para el caso de planificación estática consecutiva y realice una estimación de la aceleración que se obtendrá.

```
do i = 0, 19
      do j = 0, 199
            A(i, j) = A(i, j) + 1
            B(2i+1, j) = B(2i, j) * 2
      enddo
enddo
```

3.6.- Considere el siguiente bucle a ejecutar en una máquina paralela:

```
do i = 1, 24
      do j = 1, i
            (1) A(i,j) = B(i,j) + 1
            (2) C(i,j) = A(i-1,j) * 2
      enddo
enddo
```

a) Genere el grafo de dependencias y decide cómo paralelizar el código de la manera más eficiente posible. Escribe el código correspondiente al caso de un sistema de 4 procesadores, en el que la planificación de las tareas es dinámica por trozos de tamaño fijo, 4 iteraciones en concreto.

b) Indique el número de tareas de planificación necesarias y haz una estimación razonada del speed-up que se puede conseguir al ejecutar el programa de esa manera. Supón que el coste de ejecución de una instrucción como las del bucle es 1, y que una operación de sincronización tiene un coste de 10.

3.7.- Dado el siguiente bucle, indique cómo ejecutarlo en paralelo de la manera más eficiente posible. Escriba el código resultante. Si el tiempo de ejecución de cada instrucción es T, realice una estimación de la aceleración y de la eficiencia que se conseguirá al utilizar 10 procesadores. Justifique los resultados.

```
do i = 2, 93
        (1) A(i) = D(i) - 10
        (2) B(i+1) = C(i-1) * D(i-2)
        (3) C(i) = C(i-1) + A(i+5)
        (4) D(i-1) = E(i+1) * 5
enddo
```

3.8.- Dado el siguiente bucle, genera el grafo de dependencias e indica cómo paralelizar el código sin necesidad de sincronización (efectuando peeling.)

```
do i= 1, 198
        A(i) = B(i-1) * 3
        B(i+1) = C(i-1) + 1
        C(i) = C(i) * 0,5
        enddo
```

3.9.- Dado el siguiente bucle, genera el grafo de dependencias e indica cómo paralelizar el código sin necesidad de sincronizar los procesos.

```
do i= 1, 1000
        A(i) = A(i) +1
        B(i+1) = C(i)* A(i)
        C(i+1) = C(i+1) +D(i)
        F(i) =C(i+1)/F(i)
enddo
```

3.10.- El siguiente bucle se ejecuta en una máquina de 10 procesadores. Analiza cómo paralelizar el bucle de la manera más efectiva y escribe el código que ejecutará cada procesador para el caso de planificación estática consecutiva

Nota:cada procesador se identifica por la variable privada pid = 0..9.

```
do i = 2, 101
        do j = 0, 199
                A(i,j) = A(i,j+2) * 2
                B(i,j) = B(i-2,j) * 3
        enddo
enddo
```

3.11.- Se Se ejecuta en un sistema paralelo de 4 procesadores el siguiente bucle:

```
do i = 1, 40
     <código>
enddo
```

Todas las iteraciones son independientes entre sí y se sabe que el tiempo de ejecución de las mismas es T (i)) = i (es decir, la primera iteración 1, la segunda 2, etc.).
Calcula el tiempo de ejecución en paralelo del programa, y el factor de aceleración respecto al caso serie, si la panificación de tareas es:

a) estática: 1. consecutiva; y 2. entrelazada

b) dinámica: 1. Autoplanificación con: Z = 1 y Z = 5
2. autoplanificación guiada, Z_s = ceil [(N – i) / P]
3. trapezoidal, Z_1 = 9, k = 1

Considere que el tiempo de cada operación de planificación dinámica es 1.

3.12.- Se ejecutan en paralelo, entre 4 procesadores, 100 iteraciones de un determinado bucle, independientes entre sí.
El tiempo de ejecución de cada iteración es de 1 s, salvo una de las iteraciones, que tiene un tiempo de ejecución de 20 s.
Calcula, de manera aproximada, el speed-up y la eficiencia que se consigue en los siguientes casos:

(a) el reparto de iteraciones es estático consecutivo (con trozos del mismo tamaño).
(b) el reparto de iteraciones es estático entrelazado (con trozos del mismo tamaño).
(c) el reparto de iteraciones es dinámico, 1 a 1 (calcula el peor y el mejor caso); coste de una operación de asignación de tareas, 50 ms.

3.13.- Se quiere ejecutar en paralelo, en una máquina SMP, el bucle indicado. Escribe el código que ejecutará cada procesador, en los siguientes casos:

(a) se usan barreras como mecanismo de sincronización.
(b) se usan vectores de eventos como mecanismo sincronización.

```
do i = 4, 99
        C[i] = B[i-1] + C[i]
        A[i] = FUN1(B[i-4],
        C[i-2]) B[i] = B[i] * 0,5
enddo
```

3.14.- El siguiente código OpenMP se cuelga en tiempo de ejecución. ¿ Cual puede ser la causa?

```
/******************************************************************************
**
* FILE: omp_bug3.c
* DESCRIPTION:
* Run time error
******************************************************************************
*/
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 50
int main (int argc, char *argv[])
{
int i, nthreads, tid, section;
float a[N], b[N], c[N];
void print_results(float array[N], int tid, int section);
/* Some initializations */
for (i=0; i<N; i++)
 a[i] = b[i] = i * 1.0;
#pragma omp parallel private(c,i,tid,section)
 {
 tid = omp_get_thread_num();
 if (tid == 0)
 {
 nthreads = omp_get_num_threads();
 printf("Number of threads = %d\n", nthreads);
 }
 /*** Use barriers for clean output ***/
 #pragma omp barrier
 printf("Thread %d starting...\n",tid);
 #pragma omp barrier
 #pragma omp sections nowait
 {
 #pragma omp section
 {
 section = 1;
 for (i=0; i<N; i++)
 c[i] = a[i] * b[i];
 print_results(c, tid, section);
 }
 #pragma omp section
 {
 section = 2;
 for (i=0; i<N; i++)
 c[i] = a[i] + b[i];
 print_results(c, tid, section);
 }
 } /* end of sections */
 /*** Use barrier for clean output ***/
 #pragma omp barrier
 printf("Thread %d exiting...\n",tid);
```

```
 } /* end of parallel section */
}
void print_results(float array[N], int tid, int section)
{
 int i,j;
 j = 1;
 /*** use critical for clean output ***/
 #pragma omp critical
 {
 printf("\nThread %d did section %d. The results are:\n", tid, section);
 for (i=0; i<N; i++) {
 printf("%e ",array[i]);
 j++;
 if (j == 6) {
 printf("\n");
 j = 1;
 }
 }
 printf("\n");
 } /*** end of critical ***/
 #pragma omp barrier
 printf("Thread %d done and synchronized.\n", tid);
}
```

3.15.- El siguiente código OpenMP se cuelga en tiempo de ejecución. ¿ Cual puede ser la causa? Modifique el código para que funcione.

```
/******************************************************************************
**
* FILE: omp_bug5.c
* DESCRIPTION:
* Using SECTIONS, two threads initialize their own array and then add
* it to the other's array, however a deadlock occurs.
*
******************************************************************************
*/
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 1000000
#define PI 3.1415926535
#define DELTA .01415926535
int main (int argc, char *argv[])
{
int nthreads, tid, i;
float a[N], b[N];
omp_lock_t locka, lockb;
/* Initialize the locks */
omp_init_lock(&locka);
omp_init_lock(&lockb);
/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel shared(a, b, nthreads, locka, lockb) private(tid)
 {
 /* Obtain thread number and number of threads */
 tid = omp_get_thread_num();
 #pragma omp master
        {
        nthreads = omp_get_num_threads();
        printf("Number of threads = %d\n", nthreads);
        }
 printf("Thread %d starting...\n", tid);
 #pragma omp barrier
 #pragma omp sections nowait
        {
        #pragma omp section
        {
        printf("Thread %d initializing a[]\n",tid);
        omp_set_lock(&locka);
        for (i=0; i<N; i++)
        a[i] = i * DELTA;
        omp_set_lock(&lockb);
        printf("Thread %d adding a[] to b[]\n",tid);
        for (i=0; i<N; i++)
        b[i] += a[i];
```

```
            omp_unset_lock(&lockb);
            omp_unset_lock(&locka);
            }
            #pragma omp section
            {
            printf("Thread %d initializing b[]\n",tid);
            omp_set_lock(&lockb);
            for (i=0; i<N; i++)
            b[i] = i * PI;
            omp_set_lock(&locka);
            printf("Thread %d adding b[] to a[]\n",tid);
            for (i=0; i<N; i++)
            a[i] += b[i];
            omp_unset_lock(&locka);
            omp_unset_lock(&lockb);
            }

 } /* end of sections */
 } /* end of parallel region */
}
```