

Computación Avanzada

Memoria compartida /threads

4° Licenciatura En Sistemas de Información

2020

Universidad: UADER FCyT Concepción del Uruguay

Profesores: María Fabiana Piccoli, Carlos Casanova

Alumnos: Cepeda Leandro, Gonzalez Exequiel

Memoria compartida/threads

Conceptos Preliminares

En la actualidad, cada vez es más frecuente observar la existencia de sistemas altamente complejos que requieren mayor tiempo de computo. Para afrontar este tipo de problemas se hace uso de la computación de alto desempeño, en donde un único problema es dividido y resuelto de manera simultánea por un grupo de procesadores, donde el principal objetivo es mejorar la velocidad de procesamiento de la aplicación. En estos casos, el modelo de memoria utilizado hace que la programación de aplicaciones paralelas para cada caso sea esencialmente diferente. En este práctico estudiaremos la API de OpenMP, la cual idealmente es utilizada en aplicaciones de memoria compartida donde los procesadores se comunican a través de la memoria.

EJERCICIO 1: Hello World !!!

Desarrollar un Programa OpenMp denominado “Hola” que declare una región paralela y dentro de la misma cada thread en ejecución imprima por pantalla “Hola Mundo Desde El Thread (No thread)”. Ejecutar variando el número de threads.

tp3_ej1.cpp

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv){
    int n,tid;
    omp_set_num_threads(5);
    #pragma omp parallel private (tid, n)
    {
        tid = omp_get_thread_num();
        printf("Hola mundo desde el thread N° %d \n",tid);
    }
    return(0);
}
```

EJERCICIO 2: Repartición de tareas

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

La diferencia en la ejecución de estos programas que el programa ***tp3_ej2a.cpp*** realiza una ejecución del ciclo for normalmente, es decir, para el caso, cada proceso mostrara en pantalla el mensaje “Hola mundo %d \n”,tid” 10 veces; el programa ***tp3_ej2b.cpp*** realiza una ejecución del ciclo for dividiendo el loop entre los threads que están actualmente en ejecución, es decir, para el caso cada proceso mostrara en pantalla el mensaje “Hola mundo %d \n”,tid” 2 veces.

¿Qué sucedería si la variable n no fuera privada ?

Si la variable n no fuera privada se identificaría como compartida por todos los threads, con lo cual solo existiría una copia y todos los threads accederían y modificarían dicha copia.

tp3_ej2a.cpp

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv){
    int n,tid;
    omp_set_num_threads(5);
    #pragma omp parallel private (tid,n)
    {
        tid = omp_get_thread_num();
        for(n=0;n<10;n++){
            printf("Hola mundo %d \n",tid);
        }
    }
}
```

```
        return(0);
    }
```

tp3_ej2b.cpp

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv){
    int n,tid;
    omp_set_num_threads(5);
    #pragma omp parallel private (tid,n)
    {
        tid = omp_get_thread_num();
        #pragma omp for
        for(n=0;n<10;n++){
            printf("Hola mundo %d \n",tid);
        }
    }
    return(0);
}
```

EJERCICIO 3: Dependencia de datos.

Suponga los siguientes fragmentos de programa e indique que lazos son susceptibles de ser paralelizados y cuáles no.



Lazos susceptibles a ser paralelizados



Lazos no susceptibles a ser paralelizados

a)

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

b)

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

c)

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

d)

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

EJERCICIO 4: Suma de arreglos

Realice un programa OpenMP que realice la suma de dos arreglos componente a componente y deje el resultado en un nuevo arreglo. Ejecute el programa variando el número de threads.

```
#include <stdio.h>
#include <omp.h>
#include <iostream>

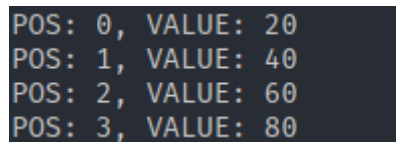
int main(int arc, char** argv)
{
    int x[] = { 10, 20, 30, 40};
    int y[] = { 10, 20, 30, 40};
    int z[] = {};
    int n;
    omp_set_num_threads(4);

    #pragma omp for
    for(n=0; n<4; n++){
        z[n] = x[n] + y[n];
    }

    for(n=0; n<4; n++)
        printf("POS: %d, VALUE: %d \n", n, z[n]);

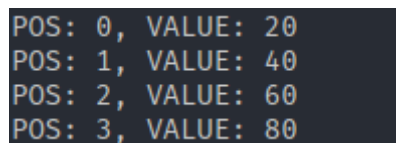
    return 0;
}
```

Ejecución con 4 threads:



```
POS: 0, VALUE: 20
POS: 1, VALUE: 40
POS: 2, VALUE: 60
POS: 3, VALUE: 80
```

Ejecución con 2 threads:



```
POS: 0, VALUE: 20
POS: 1, VALUE: 40
POS: 2, VALUE: 60
POS: 3, VALUE: 80
```

Se puede apreciar que la salida siempre será la misma, lo que varia es el tiempo de ejecución del programa.

EJERCICIO 5: Planificación de bucles

Ejecute los siguientes códigos. Describa cual es la principal diferencia entre ellos.

La principal diferencia entre ambos códigos es que el primero hace uso de la programación estática, en la cual los bloques de iteraciones se asignan estáticamente a los subprocesos de ejecución en forma de turnos. Lo bueno de la programación estática es que el tiempo de ejecución de OpenMP garantiza que si tiene dos bucles separados con el mismo número de iteraciones y los ejecuta con el mismo número de subprocesos utilizando la programación estática, entonces cada subproceso recibirá exactamente el mismo rango de iteraciones (s) en ambas regiones paralelas; y el segundo hace uso de la programación dinámica, la cual funciona según el orden de llegada. En este tipo de programación puede pasar que dos ejecuciones con la misma cantidad de subprocesos podrían (y probablemente lo harían) producir asignaciones de "espacio de iteración" -> "subprocesos" completamente diferentes

tp3_ej5a.cpp

```
#include <omp.h>
#include <stdio.h>
#include <unistd.h>

#define N 40
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");

    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");
}
```

tp3_ej5b.cpp

```
#include <omp.h>
#include <stdio.h>
#include <unistd.h>
#define N 40

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");
```

```
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");
```

```
}
```