Práctica 1: Suma Paralela

Muñiz Patiño, Andrea Fernanda.
Versión 3.1

1. Objetivo

Implementar la suma de los primeros 1 000 000 000 números naturales, eventualmente realizar un análisis de como afecta el uso de los procesadores en el rendimiento del algoritmo.

2. Actividad para realizar en laboratorio

Suma de los primeros *1 000 000 000* números naturales de forma paralela, además de medir el tiempo de ejecución para realizar la suma.

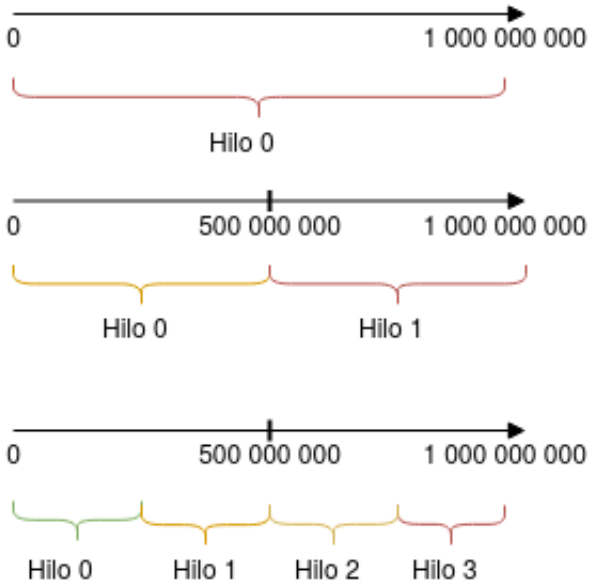


Figura 1: Esquema de suma paralela de los primeros 1 000 000 000 números

Como se muestra en la Figura 1, queremos que cada hilo haga la suma en paralelo de un intervalo $[n, m]$ el cual va a estar determinado por el *identificador* del hilo que está trabajando. Por ello es necesario que cada hilo tenga acceso a una variable común llamada *sumaTotal*.

Un punto importante ha recordar de los algoritmos paralelizados es que el desempeño computacional dependerá de los parámetros de entrada y el número de procesadores.

3. Parámetros y tiempo en C

Queremos que al momento de ejecutar nuestro algoritmo, este reciba como parámetro la cantidad de hilos que llevarán acabo la tarea. Esto con la finalidad de realizar de la siguiente manera la ejecución de nuestra actividad.

```
umm@usuario:~$ ./a.out  n
```

Donde **n** es el número de hilos que queremos pasar como parámetro.

Esto se logra con el siguiente código.

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

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

//verifica el numero de hilos que se paso como parametro
        if (argc < 2){
                printf("por favor especifique el numero de hilos\n");
                exit(1);
        }

}
```

Recuerda no aceptar un número de hilos inválido. Por ejemplo no tiene sentido tener $-n$ hilos, donde n es un número entero.

Para obtener el tiempo de ejecución del programa podemos hacerlo con *struct* timeval, así como se muestra en el siguiente código.

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <omp.h>
/* Codigo para obtener el tiempo que toma la ejecucion de un programa
*       Struct timeval
*/
int main(int argc, char** argv){
        int suma;
        struct timeval inicio, fin;//nos permiten medir el tiempo de ejecucion

        gettimeofday(&inicio, NULL);//guarda el tiempo al inicio del programa
        int tiempo;

        for(int i =0; i < 10; i++){
                suma = suma +1;
        }

        gettimeofday(&fin, NULL); //guarda el tiempo al final del programa
        tiempo = (fin.tv_sec - inicio.tv_sec)* 1000000 + (fin.tv_usec - inicio.
            tv_usec);
        printf("suma: %i  \ntiempo de ejecucion: %i microsegundos\n",suma, tiempo);
            //imprime resultados
}
```

4. Observaciones

4.1. Desarrollo en laboratorio

Las siguientes preguntas deben hacerse en la clase de laboratorio para que el alumno pueda realizar un análisis previo a la actividad.

- ¿Está relacionado el número de núcleos y procesadores, con el número de hilos que ejecuta un programa?
- ¿Qué arquitectura estamos utilizando para esta actividad?

Respuesta: SIMD (Single Instruction Multiple Data) y CRCW (Concurrent Read Concurrent Write)

5. Actividad para el alumno

Con ayuda del código implementado en el laboratorio, deberás implementar la suma de los primeros 1 000 000 000 de forma paralela, la cual será capaz de recibir desde terminal cuantos hilos creará el programa ya que deberás usar el identificador del hilo en la implementación de la suma, además deberás realizar un experimento práctico que consiste en ejecutar la solución paralela en varias ocasiones y con diferente número de hilos.

Ejecutarás en una terminal ***A*** la actividad realizada en clase y obtendrás los tiempos de cada ejecución para presentar los resultados en una tabla con el siguiente formato:

No. de Hilos	Tiempo de ejecución T(p)	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5
1						
2						
4						
6						
20						
50						
100						

Cuadro 1: Tabla de tiempo de ejecución

La celda **Tiempo de Ejecución T(p)** es el promedio de las cinco pruebas para el número de hilos correspondiente para el renglón.

Después abrirás una terminal ***B*** donde compilarás y pondrás en ejecución el código que se te proporcionó para está práctica, llamado **cod01.c** y volverás a obtener los valores de *Tiempo de ejecución T(p)* y los reportaras de la misma manera que la tabla anterior, especificando que las pruebas se realizaron con el programa **cod01.c** en la Terminal **B** y el programa que suma los primeros 1 000 000 000 números en la Terminal ***A***.

5.1. Código cod01.c

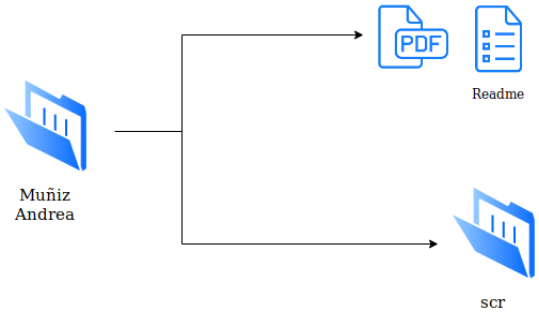
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <omp.h>

int main(int argc, char** argv){
        int numHilos, idHilo;

        omp_set_num_threads(100);
        //inicia seccion paralela
        #pragma omp parallel private(idHilo)
    {
                while(1){
                        idHilo = omp_get_thread_num();
                        printf(" Que  hago?  Que  estoy provocando? %i \n", idHilo);
                }
        }//fin de seccion paralela
}
```

5.2. Archivos a entregar

- Para la entrega de esta práctica deberás crear una carpeta con tu nombre y apellido, en ella pondrás un archivo *readme*, donde están las especificaciones sobre el programa y el ***PDF*** correspondiente, una sub-carpeta llamada *src* en la cual estarán los códigos fuente.



- Como parte de la Actividad 2, deberás entregar un reporte con los siguientes requisitos y contestando a las preguntas:
  - Anexar las dos tablas de tiempos obtenidas.
  - Añadir en el reporte la información sobre tu equipo de cómputo que obtuviste en la Actividad 1, es decir, el número de núcleos y procesadores. ¿Cómo afecta el número de procesadores y núcleos en el Cuadro 1 que presentaste?
  - Explicar cómo afecta el programa **cod01.c** al desempeño de tu algoritmo y cuál es el objetivo del código **cod01.c**.

5.3. Apoyo

Si usas latex para tus reportes aquí te proporciono el código de la tabla para el reporte, basta con que llenes la tabla con los valores obtenidos en tu experimento práctico.

```
\begin{table}[h]
    \begin{tabular}{ |c|c|c|c|c|c|c|}
     \hline
    No. de Hilos & Tiempo de ejecuci n T(p) & Prueba 1 & Prueba 2 & Prueba 3 & Prueba 4 & Prueba 5\\
    \hline
    1 & &&&& \\
    \hline
    2 & &&&&\\
    \hline
    6 & &&&&\\
    \hline
    8 & &&&&\\
    \hline
    20 & &&&&\\
    \hline
    50 & &&&&\\
    \hline
    100 & &&&&\\
    \hline
    \end{tabular}
        \caption{Tabla de tiempo de ejecuci n}
\label{table:1}
\end{table}
```

Otro recurso disponible es: https://www.tablesgenerator.com/

6. Solución

A continuación se presenta una solución para el problema propuesto para la actividad de laboratorio.

Código que realiza la suma en paralelo.

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <omp.h>

/*
*Realiza la suma de los primeros 1,000,000,000 numeros naturales de forma paralela
*y lleva registro del tiempo de ejecucion
*/


int main(int argc, char** argv){
        struct timeval inicio, fin;//nos permiten medir el tiempo de ejecucion
        gettimeofday(&inicio, NULL);//guarda el tiempo al inicio del programa
        int tiempo;

        int MAX_HILOS=100; //limite superior (arbitrario) del numero de hilos
        int numHilos, idHilo;

        //verifica el numero de hilos que se paso como parametro
        if (argc < 2){
                printf("por favor especifique el numero de hilos\n");
                exit(1);
        }
        sscanf(argv[1], "%i", &numHilos);
        if (numHilos < 1 || numHilos > MAX_HILOS){
                printf("Numero de hilos no valido (%i)\n", numHilos);
                exit(1);
        }

        omp_set_num_threads(numHilos);
        long int suma, sumaHilo;
        suma=0;

        //inicia seccion paralela
        #pragma omp parallel private(idHilo, sumaHilo)
    {
                idHilo = omp_get_thread_num();
                if (idHilo==0){
                        printf ("iniciando calculo con %i hilos\n",
                            omp_get_num_threads());
                }

                //cada hilo realiza una suma parcial
                sumaHilo=0;
                int i;
                for (i=idHilo;i<=1 000 000 000;i+=numHilos){
                        sumaHilo+=i;
                }

                //cada hilo actualiza la suma total con su resultado parcial
                suma+=sumaHilo;
        }//fin de seccion paralela

        gettimeofday(&fin, NULL); //guarda el tiempo al final del programa

```

```
56          tiempo = (fin.tv_sec - inicio.tv_sec)* 1000000 + (fin.tv_usec - inicio.
                tv_usec);
57
58          printf("suma: %li  \ntiempo de ejecucion: %i microsegundos\n",suma, tiempo);
                //imprime resultados
59  }
```

Información sobre el equipo de cómputo donde se ejecutó la actividad.

```
ummyers@ummyers:~$ nproc --all
4
ummyers@ummyers:~$ lscpu | grep 'CPU(s)'
CPU(s):                          4
Lista de la(s) CPU(s) en línea:  0-3
CPU(s) del nodo NUMA 0:          0-3
ummyers@ummyers:~$
```

La siguiente tabla presenta los valores correspondientes obtenidos cuando se ejecuta la solución propuesta anteriormente. Los tiempos están en microsegundos.

No. de Hilos	Tiempo de ejecución T(p)	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5
1	2221686,2	2216924	2233832	2230004	2227005	2200666
2	1212596	1206060	1215533	1166043	1223968	1251376
4	1107980,8	1069859	1071751	1122743	1084856	1190695
6	1145813,6	1089914	1124471	1180721	1150418	1183544
20	1131820,8	1075369	1131635	1169270	1154892	1127938
50	1133403,8	1081589	1155831	1142450	1136843	115030
100	1090661,2	1088272	1085359	1077672	1117955	1084048

Cuadro 2: Solución

La siguiente tabla tiene los resultados cuando se está ejecutando el Código **cod01.c** en la Terminal **B** y la suma en paralelo en la Terminal **A**.

No. de Hilos	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5
1	2640981	2565946	2631623	2596611	2614309
2	1587423	1451607	1670709	2004711	1988991
4	1354638	1415041	1361970	1323570	1401350
6	1426965	1317964	1403669	1353530	1316549
20	1264439	1229630	1241255	1207024	1237820
50	1194481	1203597	1192135	1164559	1165223
100	1229760	1216745	1226873	1216220	1224791

Cuadro 3: Tabla de tiempo de ejecución, con recursos compartido

6.1. Breve análisis

El propósito de estar ejecutando el programa **cod01.c** en otra terminal y luego ejecutar la suma paralela se hace con el fin de observar en microsegundos el impacto que tiene el estar usando los procesadores con una tarea tan sencilla como cien hilos imprimiendo en pantalla indefinidamente cuando queremos resolver alguna otra tarea.

Observemos que en el Cuadro 2 el mejor tiempo $(1107980,8\mu s)$ que se presenta es cuando el programa usa únicamente 4 hilos, la computadora donde se ejecutaron las pruebas del cuadro 2, es una DELL con 4 CPU(s), esto es por qué cada hilo fue designado a cada CPU. En el Cuadro 3, el cual presenta los resultados de los tiempos obtenidos cuando se usan los CPU(s) para estar imprimiendo en pantalla y a la vez hacer la suma de los primeros 1 000 000 000 números, tenemos que los mejores tiempos obtenidos corresponden al renglón de las pruebas realizadas con 50 hilos. Incluso en el Cuadro 3 podemos observar que los mejores tiempos corresponden

a las pruebas realizadas con 50 hilos. Recordemos que cualquier problema tiene una cota en tiempo óptimo, además también hay una cota para el número de procesadores que pueden optimizar un proceso. Cada problema tendrá un número de hilos óptimo, para este caso tenemos que 50 hilos nos proporciona el tiempo óptimo, al usar 100 aumentamos el tiempo en vez de disminuirlo, se tendría que buscar el número n donde se pierde la optimización del problema. Sin olvidar claro que esta compartiendo recursos con el código **B**, por lo cual si el Código **B** se cambia o modifica es probable que el número óptimo de hilos también cambie.

No. de Hilos	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5
1	2640981	2565946	2631623	2596611	2614309
2	1587423	1451607	1670709	2004711	1988991
4	1354638	1415041	1361970	1323570	1401350
6	1426965	1317964	1403669	1353530	1316549
20	1264439	1229630	1241255	1207024	1237820
50	1194481	1203597	1192135	1164559	1165223
100	1229760	1216745	1226873	1216220	1224791

Cuadro 4: Tabla de tiempo de ejecución, con recursos compartidos

En el Código **cod01.c** estamos usando 100 hilos para saturar los procesadores e imprimir en pantalla, lo cual ocasiona que los tiempos, cuando se crean 4 hilos para hacer la suma en paralelo, no sean los óptimos, esto debido a que se comparten recursos. Recordemos que el equipo donde se ejecutó el código tiene 4 procesadores, esto significa que se está compartiendo tiempo de procesador, ya que tenemos 104 hilos ejecutándose de manera paralela y al ser *OpenMP* una simulación tenemos que los 104 hilos están compartiendo tiempo de ejecución en los procesadores.

6.1.1. Información adicional del equipo de cómputo

Con el fin de comprender mejor los resultados obtenidos en las tablas anteriores se presenta más información sobre la arquitectura de la computadora donde se realizaron las pruebas.

Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz

La anterior información se obtuvo de ejecutar el siguiente comando:

```
umm@usuario:~$ sudo lshw
```

Además también podemos obtener información adicional sobre cada *core* con el siguiente comando:

```
umm@usuario:~$ cat proc/cpuinfo
```