Informe Trabajo Final

Programación concurrente

Docente: Maximiliano A. Eschoyez.

Alumnos: Bobadilla Barcelo Daniel Agustin.

Genaro Kevin Luis.

Institución: Universidad Blas Pascal.

Carrera: Ingeniería Informática.

Materia: Programación concurrente.

Fecha de entrega: 23/07/2021.

Consigna

Resumen

El objetivo de este Trabajo Final es realizar un software para integración numérica utilizando técnicas de programación paralela. Se deberá implementar el sistema utilizando la Interfaz de Paso de Mensajes — MPI y la interfaz de hilos paralelos OpenMP. Además, se debe realizar una serie de mediciones de tiempo de ejecución para comparar el programa secuencial, el comportamiento del programa paralelizado con múltiples hilos y con múltiples procesos.

Consigna principal

En este trabajo se deben desarrollar dos versiones de un programa, tanto para su implementación con la biblioteca OpenMP y con MPI (son 4 programas en total), para realizar la integración numérica de diferentes funciones.
El objetivo es dividir el espacio de integración entre todos los procesos/hilos que se vayan a ejecutar. De esta forma, la cantidad de procesos/hilos dependerá de los recursos disponibles o la configuración de inicio, y cada proceso buscará sólo en el rango que le corresponde.
La primer versión del programa debe ejecutar cada método de integración en un hilo diferente para OpenMP o en un proceso para MPI.
La segunda versión debe dividir la tarea de integración de cada método en n hilos OpenMP o procesos MPI y, además, hacer que se ejecuten los cuatro métodos de integración en paralelo. Es decir, si n = 4 se tienen que ejecutar los 4 métodos con 4 hilos/procesos cada uno, totalizando 16 hilos/procesos en la CPU.
Los métodos de integración a implementar son:
1. Regla del Rectángulo,
2. Regla del Punto Medio,
3. Regla del Trapecio,
4. Simpson 1/3.

Se sugiere utilizar funciones sencillas como f (x) = x y polinomios de diverso orden para poder verificar la correcta implementación de las fórmulas. Luego, probar con funciones más complejas y gran cantidad de intervalos de integración.

Presentación del Trabajo Final

Grupos de Trabajo

El trabajo se podrá presentar en forma individual o en grupo de dos integrantes, prefiriéndose la modalidad grupal.

Código Fuente

El código fuente y la versión digital del informe en PDF deben entregarse a través del enlace correspondiente en la plataforma MiUBP del examen final (http://mi.ubp.edu.ar/). En

dicho enlace se deberá subir un único archivo en formato ZIP conteniendo todos los código fuente que se requieran para la realización del trabajo final.

Informe Escrito
Se entregará al profesor un informe escrito en versión digital donde se debe describir la problemática abordada en el trabajo final, el desarrollo de la solución propuesta, los resultados de las mediciones de tiempo y una conclusión. El texto deberá ser conciso y con descripciones apropiadas. No se debe incluir el código fuente, sino los textos necesarios para realizar las explicaciones pertinentes. El formato de entrega es PDF.

Explicación de las versiones

Versiones de los programas :

● Final_Secuencial.c : Esta versión muestra de forma secuencial como integrar una función dada en un determinado intervalo con los métodos mencionados en la consigna. Básicamente hicimos una función por cada método de integración que se llaman desde el main y nos dan el resultado de la integral para cada uno de los métodos.

● Final_OMP1.c : En la primera versión usando la librería OpenMP utilizamos nuevamente las funciones de los metodos de integracion pero esta vez cada método es calculado por un hilo diferente. Para ello utilizamos las directivas pre-procesador #pragma omp parallel para paralelizar nuestro programa y luego #pragma omp sections en donde creamos cuatro secciones diferentes, cada hilo ingresa a una única sección y realiza el cálculo de la integral correspondiente.

● Final_OMP2.c : En la segunda versión de OpenMP nuevamente cada hilo se encarga de un método de integración específico pero a su vez se divide en cuatro hilos repartiendo la tarea entre estos. La función omp_set_nested(1) nos permite el paralelismo anidado. Cabe aclarar que dentro de cada función de cada método se hace esta división de las tareas con la función omp_set_num_threads(4) y con la directiva #pragma for schedule a cada hilo se le da una porción(chunk) para integrar.

● Final_MPI1.c : En la primera versión usando la librería MPI utilizamos nuevamente las funciones de los metodos de integracion y cada método es calculado por un proceso diferente. Para sincronizar los distintos procesos inicializamos el entorno de ejecución MPI con MPI_Init() y a la hora de ejecutar especificamos el número de procesos, en este caso cuatro procesos. El proceso root se encarga de un método ,

los demás calculan el resto y envían al root el resultado correspondiente. Esto se logra gracias a MPI_Recv(...) y MPI_Send(...). Al terminar se cierra el entorno con MPI_Finalize().

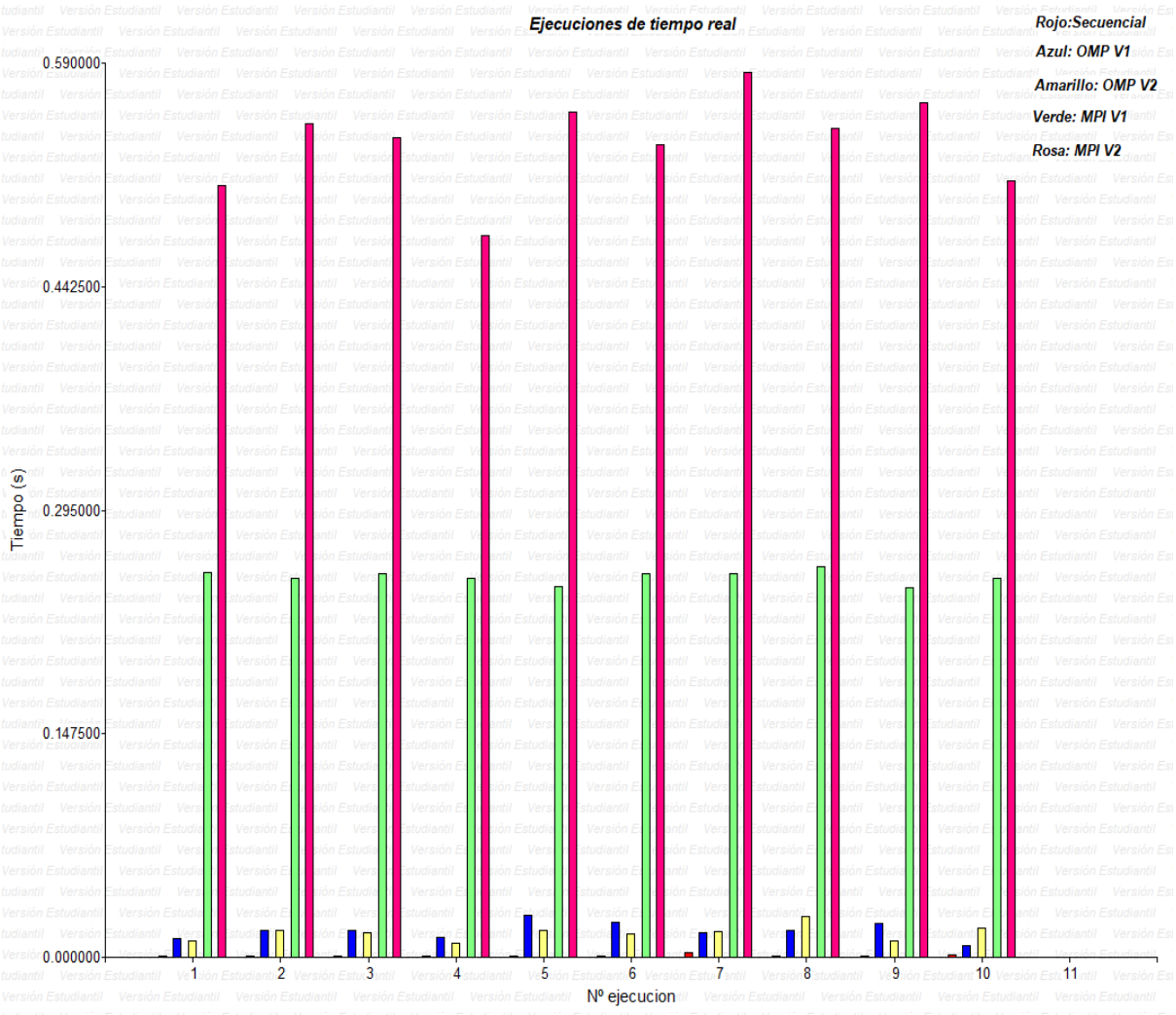
● Final_MPI2.c : En la segunda versión utilizando MPI nuevamente cada proceso se encarga de un metodo de integracion pero a su vez se subdivide en cuatro procesos para realizar la tarea. Nuevamente inicializamos el entorno MPI con MPI_Init() y al momento de ejecutar especificamos 16 procesos. Una vez se sincronizan , hacemos una división de los mismos en cuatro grupos diferentes utilizando MPI_Comm_split(...) creando un segundo comunicador. A cada grupo se le asigna cuatro procesos que realizan el cálculo correspondiente. Al final del programa se pone una barrera con MPI_Barrier(...) para que todos se sincronicen y el proceso root cierre el entorno MPI.

Pruebas de ejecución

Utilizamos la función f(x) = x ² en el intervalo [3,30] con una cantidad de 12800 subintervalos y se realizaron 10 ejecuciones por versión. Estos fueron los tiempos de ejecución total y tiempo de uso de CPU:

Versión / Time (s)	Secuencial Total	Secuencial CPU	OMP v1 Total	OMP v1 CPU	OMP v2 Total	OMP v2 CPU	MPI v1 Total	MPI v1 CPU	MPI v2 Total	MPI v2 CPU
1	0.000842	0.000680	0.012558	0.032549	0.010740	0.023774	0.253594	0.000659	0.508301	0.007212
2	0.001050	0.000745	0.017738	0.029545	0.017414	0.032637	0.249757	0.000586	0.550125	0.012880
3	0.001101	0.000800	0.017595	0.036449	0.016517	0.035863	0.253343	0.000835	0.540793	0.009607
4	0.000785	0.000700	0.013196	0.028454	0.009165	0.020278	0.250368	0.000397	0.475856	0.007478
5	0.001	0.000	0.027	0.028	0.017	0.046	0.244	0.000	0.557	0.009

	103	973	342	196	791	383	376	201	729	823
6	0.000 970	0.000 784	0.023 401	0.039 983	0.015 494	0.040 655	0.253 414	0.000 382	0.536 364	0.005 790
7	0.002 867	0.002 315	0.016 245	0.037 483	0.016 574	0.040 556	0.253 3	0.001 542	0.583 750	0.007 541
8	0.000 787	0.000 702	0.017 849	0.039 038	0.026 743	0.047 519	0.257 566	0.000 732	0.546 628	0.005 494
9	0.000 748	0.000 657	0.021 952	0.040 462	0.010 890	0.021 401	0.241 613	0.000 381	0.563 601	0.006 989
10	0.001 321	0.001 182	0.007 748	0.016 105	0.019 263	0.042 523	0.249 796	0.000 952	0.512 448	0.003 013



Conclusiones

Elegimos la función f(x) = x ², intervalo [3,30] y con una cantidad de subintervalos de 12800. En todas las versiones obtenemos los mismos resultados solo que cambian los tiempos de ejecución total del programa y el uso de CPU.
Como se puede observar en el gráfico el tiempo de ejecución de la versión Secuencial es menor a las versiones utilizando OpenMP y MPI. Esto se debe a que al ser métodos bastantes sencillos el procesador puede realizarlos con un hilo/procesador. Si agregamos hilos/procesos en paralelo el tiempo de ejecución es mayor ya que el procesador debe asignarles las distintas tareas a cada uno y recolectar los distintos resultados.
En cuanto, al uso de OpenMP y MPI podemos ver que trabajar con hilos en OpenMP nos muestra ejecuciones de menor tiempo ya que la comunicación entre hilos al compartir memoria y recursos no necesita la invocación del núcleo del sistema operativo. Otro motivo es que se tarda mucho menos tiempo en crear un hilo que un proceso y también en conmutar entre ellos.