Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
Computación Paralela

Proyecto 1

Repositorio:
https://github.com/JuanPineda115/proyecto1-Paralela

Diego Crespo 19541
Ale Gudiel 19232
Juan Pablo Pineda 19087
Gabriel Quiroz 19255
José Pablo Ponce 19092

INDICE

Introducción

En la era de la computación moderna, el aprovechamiento eficiente de los recursos computacionales, como los procesadores multinúcleo, es crucial para el rendimiento y la eficiencia de las aplicaciones. La programación paralela es una técnica ampliamente utilizada para mejorar el rendimiento y la eficiencia en el procesamiento de datos, al dividir el trabajo en múltiples tareas que se ejecutan simultáneamente en diferentes núcleos o procesadores.

El proyecto que se presenta en este documento tiene como objetivo principal desarrollar un "screensaver" paralelo utilizando OpenMP, una herramienta que facilita la transformación de programas secuenciales en paralelos y permite la abstracción y programación paralela en alto nivel.

Se comenzará diseñando e implementando un algoritmo secuencial para resolver un problema que tiene potencial para ser paralelizado. Luego, se modificará y mejorará el programa buscando un incremento en el desempeño del mismo, basándose en los conceptos de speedup y eficiencia estudiados.

El screensaver desarrollado debe recibir como mínimo un parámetro (N), que indica la cantidad de elementos a renderizar, y debe desplegar varios colores, idealmente generados de forma pseudoaleatoria. Además, el tamaño del canvas debe ser de mínimo 640x480 (w,h), y el screensaver debe tener movimiento y ser estéticamente interesante, incorporando algún elemento de física o trigonometría en sus cálculos. Por último, se debe desplegar la cantidad de FPS (frames per second) en los que corre el programa para garantizar una experiencia de usuario adecuada. El resultado final será un screensaver paralelo eficiente y visualmente atractivo.

Anexo 1

Diagrama de Flujo:

3. Se declaran las variables globales que se utilizarán en el programa, incluyendo las posiciones y velocidades iniciales de las pelotas, el número de pelotas y los colores de cada pelota.

2. Se define una función para generar un color aleatorio.

3. Se define una función de dibujo que realiza los siguientes pasos:

a. Incrementa un contador de frames.

b. Obtiene el tiempo actual en milisegundos.

c. Calcula el tiempo transcurrido desde la última actualización de FPS.

d. Actualiza el FPS si ha pasado un segundo desde la última actualización.

e. Borra la pantalla.

f. Dibuja cada pelota en su posición actual con su respectivo color.

g. Muestra el FPS en la pantalla.

h. Intercambia los buffers.

i. Usa OpenMP para paralelizar el cálculo de la posición de las pelotas:

4. Se define una función de inicialización de OpenGL que realiza los siguientes pasos:

a. Genera posiciones y velocidades aleatorias para cada pelota.

El programa principal llama a la función de inicialización de OpenGL.

El programa principal entra en un bucle de eventos de GLUT que realiza los siguientes pasos en cada iteración:

a. Llama a la función de dibujo.

b. Espera por eventos de entrada.

c. Procesa eventos de entrada si hay alguno.

- Captura de argumentos

  Captura de argumentos: No hay captura de argumentos personalizada desde la línea de comandos. El programa utiliza glutInit(&argc, argv); para pasar los argumentos de línea de comandos al sistema de ventana GLUT, pero no hay otros argumentos capturados para uso personalizado en el programa.

- Solicitud de ingreso de datos

  El código no solicita ningún dato de entrada al usuario, ya que las posiciones, velocidades y colores de las pelotas se generan aleatoriamente utilizando números aleatorios y no dependen de la entrada del usuario.

- Programación defensiva

  El código paralelo cuando no se le indica el parámetro de hilos al momento de su ejecución no se inicializa y se evitan errores al momento de su ejecución.

- Secciones paralelas

  La función "private" se utiliza para crear una variable local para cada hilo que se ejecuta en la sección paralela. Esto garantiza que cada hilo tenga su propia copia de la variable, lo que evita conflictos de memoria y mejora el rendimiento.
  La función "atomic" se utiliza para garantizar que una variable sea actualizada por un solo hilo a la vez. Esto es importante cuando se trabaja con variables que se actualizan frecuentemente, ya que puede haber conflictos de memoria y errores en la ejecución del programa si varios hilos intentan actualizar la misma variable al mismo tiempo.
  La función "parallel for" se utiliza para ejecutar un bucle en paralelo, lo que permite que varias iteraciones del bucle se ejecuten simultáneamente en diferentes hilos. Esto puede mejorar significativamente el rendimiento del programa, especialmente si el bucle tiene un gran número de iteraciones.

- Mecanismos de sincronía

  OpenMP llamado "parallel for" para paralelizar el cálculo de la posición de las pelotas

- Despliegue de resultados

El código muestra el "screensaver" con múltiples pelotas que rebotan en la ventana de OpenGL. Las pelotas tienen colores aleatorios y velocidades aleatorias.
El programa también muestra los fotogramas por segundo (FPS) en la esquina inferior izquierda de la ventana de OpenGL.
Al final de la ejecución, el programa imprime el tiempo transcurrido en segundos en la consola.

Anexo 2

- Entradas − nombres de variables, tipos y descripción de su uso
    - screenWidth (int) - Ancho de la ventana de OpenGL.
    - screenHeight (int) - Altura de la ventana de OpenGL.
    - ballRadius (double) - Radio de las pelotas que se dibujan.
    - N (const int) - Cantidad de pelotas que se dibujan.
    - ballColor[N][3] (float) - Colores de las pelotas (valores RGB).

- Salidas − nombres de variables, tipos y descripción de su uso
    - ballX[N] (double) - Posiciones X de las pelotas.
    - ballY[N] (double) - Posiciones Y de las pelotas.
    - ballVelocityX[N] (double) - Velocidades X de las pelotas.
    - ballVelocityY[N] (double) - Velocidades Y de las pelotas.
    - fps (int) - Fotogramas por segundo (FPS) mostrados en la ventana.

- Descripción − Propósito de la función / clase / subrutina y descripción del funcionamiento

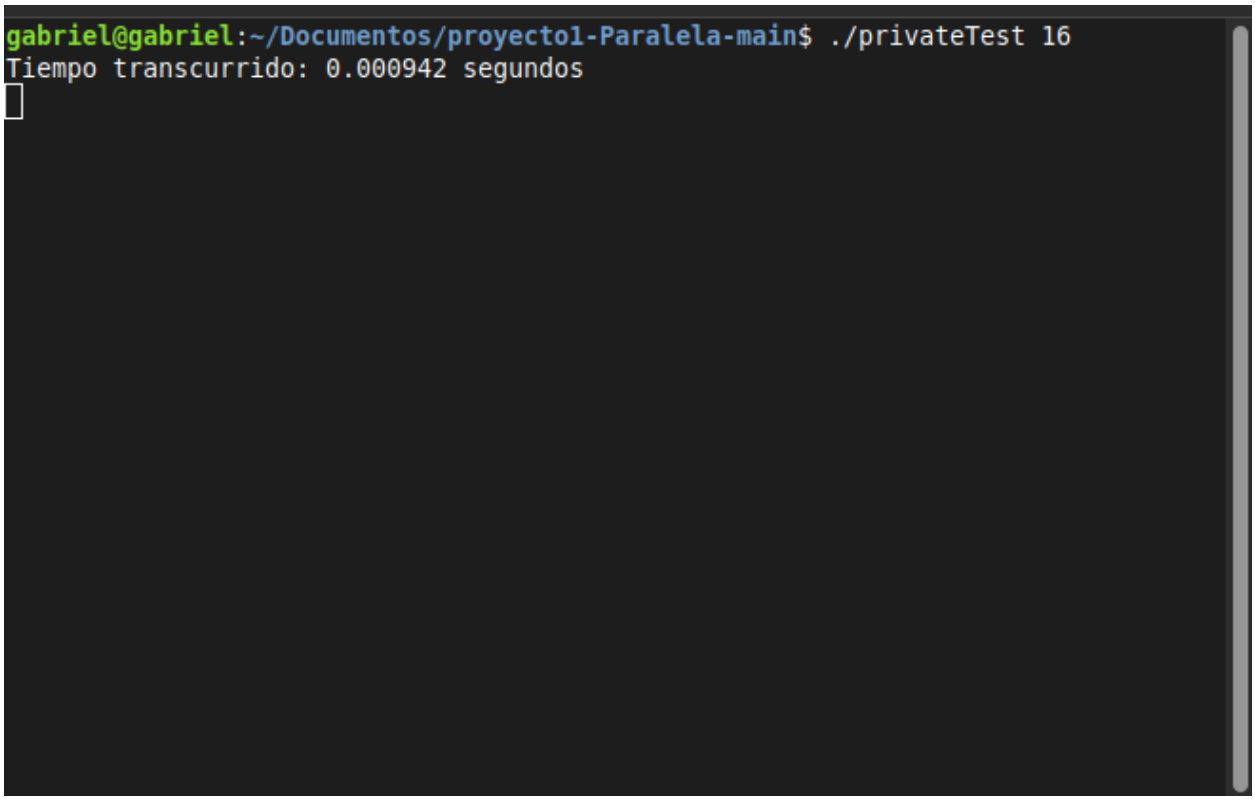
    - El propósito de este programa es crear un "screensaver" con múltiples pelotas que rebotan en una ventana de OpenGL. Las pelotas tienen colores aleatorios y velocidades aleatorias.
      Funcionamiento: El programa utiliza la biblioteca GLUT para manejar la ventana de OpenGL y el ciclo de eventos. Cuando se ejecuta, se crea una ventana de OpenGL y se dibujan varias pelotas que rebotan en ella. Las pelotas tienen colores aleatorios y velocidades aleatorias. Además, el programa muestra los fotogramas por segundo (FPS) en la esquina inferior izquierda de la ventana de OpenGL. Al final de la ejecución, el programa imprime el tiempo transcurrido en segundos en la consola.

Anexo 3

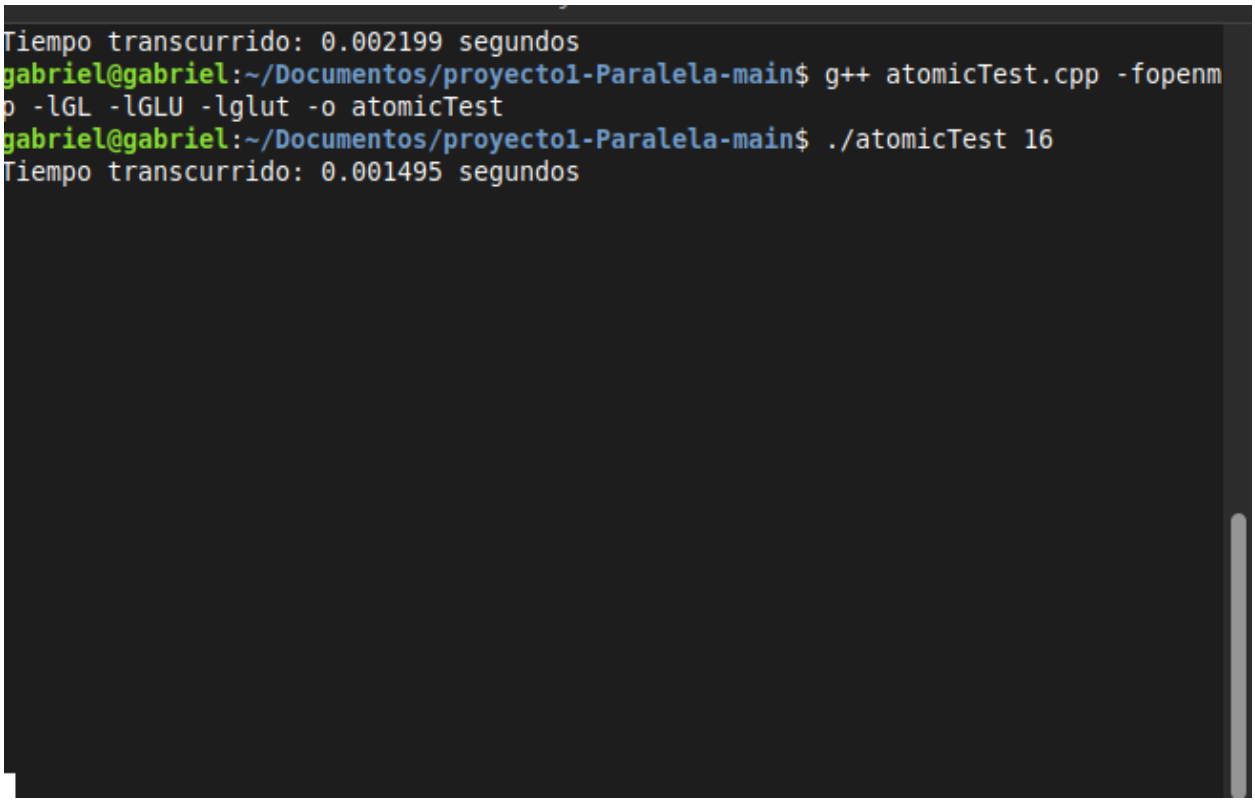
PARALLEL FOR

```
gabriel@gabriel:~/Documentos/proyecto1-Paralela-main$ g++ forTest.cpp -fopenmp -
lGL -lGLU -lglut -o forTest
gabriel@gabriel:~/Documentos/proyecto1-Paralela-main$ ./forTest 16
Tiempo transcurrido: 0.001335 segundos
```

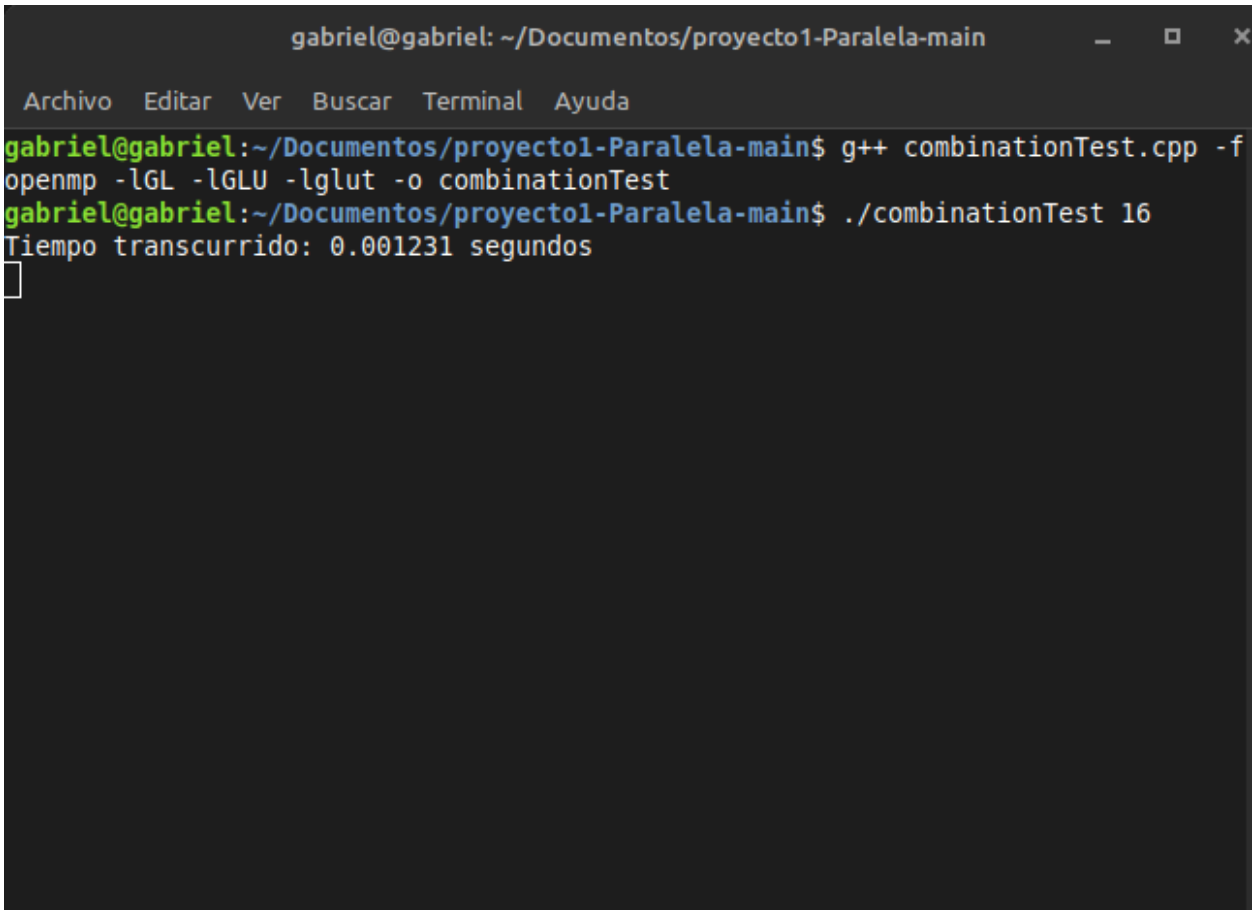
PRIVATE



ATOMIC



COMBINATION OF ALL



	Secuencial	Paralelo (Combination ALL)	paralelo(Private)	paralelo3 (Atomic & Parallel For)	PARALEL (Parallel For)
	0.002305	0.001231	0.00149	0.001495	0.001335
	0.002349	0.001121	0.002874	0.001395	0.001192
	0.002313	0.00148	0.002114	0.001908	0.002039
	0.002466	0.001753	0.0011	0.001798	0.001668
	0.002214	0.001384	0.0023	0.001675	0.002001
	0.002134	0.001274	0.002651	0.001163	0.001735
	0.001923	0.001184	0.002745	0.001863	0.001184
	0.002185	0.001562	0.00148	0.001114	0.001863
	0.002034	0.001139	0.00232	0.001105	0.001678
	0.001919	0.001261	0.001041	0.001278	0.001349
promedio	0.0021842	0.0013389	0.0020115	0.0014794	0.0016044
speedup	N/A	1.631339159	1.085856326	1.476409355	1.253739716
eficiencia	N/A	0.1019586974	0.06786602038	0.0922755847	0.07835873224

Discusión de Resultados

La utilización de técnicas de paralelismo es común en la programación moderna para mejorar el rendimiento de los programas al permitir la ejecución simultánea de múltiples tareas. En este proyecto, se han utilizado diferentes funciones de paralelismo para generar 100 pelotas con hilos y se ha evaluado el tiempo que tarda cada una en completar la tarea.

Los resultados indican que la combinación de varias funciones de paralelismo puede mejorar significativamente el rendimiento del programa, lo que se evidencia en el tiempo que tarda en completar la tarea. Específicamente, se observó que el uso de la función "parallel for" y la combinación de varias funciones de paralelismo resultó en un tiempo significativamente menor en comparación con los otros códigos evaluados. La función "parallel for" permite la ejecución simultánea de varias iteraciones de un bucle, lo que acelera significativamente el proceso.

Por otro lado, se encontró que el uso de la función "atomic" resultó en un tiempo más largo en comparación con las otras funciones de paralelismo evaluadas. Esto se debe a que "atomic" garantiza que una variable sea actualizada por un solo hilo a la vez, lo que puede ralentizar el proceso.

En conclusión, los resultados sugieren que el uso de múltiples funciones de paralelismo y la función "parallel for" pueden mejorar significativamente el rendimiento del programa. Sin embargo, es importante considerar que la elección de la función de paralelismo adecuada dependerá del tipo de tarea que se esté realizando y del hardware en el que se esté ejecutando el programa. Estos resultados podrían ser relevantes para desarrolladores que buscan optimizar el rendimiento de sus programas mediante la implementación de técnicas de paralelismo.

Recomendaciones

- Estudie OpenMP y OpenGL: Asegúrese de comprender los conceptos básicos de la programación paralela utilizando OpenMP y el manejo de gráficos mediante OpenGL. Puede comenzar por leer documentación oficial, tutoriales y ejemplos de código en línea.

- Configure su entorno de desarrollo: Instale las herramientas y bibliotecas necesarias, como compiladores compatibles con OpenMP y bibliotecas de OpenGL. Asegúrese de que su entorno de desarrollo sea compatible con estas tecnologías.

- Comprenda el código proporcionado: Estudie el código proporcionado en este proyecto y asegúrese de comprender cada sección. Preste atención a cómo se

utiliza OpenMP para paralelizar ciertas partes del código y cómo OpenGL se utiliza para el manejo de gráficos.

- Adapte el código a sus necesidades: Si desea modificar el código para abordar un problema diferente o agregar nuevas funciones, identifique las secciones relevantes del código y realice las modificaciones necesarias.

- Pruebe y optimice su solución: Ejecute su versión modificada del proyecto y evalúe su rendimiento utilizando métricas como el speedup y la eficiencia. Realice ajustes iterativos en el código para mejorar el rendimiento y la eficiencia.

Conclusiones

- Se ha logrado diseñar e implementar un algoritmo secuencial para crear un screensaver con elementos gráficos en movimiento. Este screensaver ha sido posteriormente paralelizado mediante la herramienta OpenMP, lo cual ha permitido un aumento en el desempeño y eficiencia del programa.

- El empleo del método PCAM y los conceptos de patrones de descomposición y programación han sido fundamentales para transformar el programa secuencial en uno paralelo. Gracias a esto, se ha conseguido aprovechar los múltiples recursos disponibles y mejorar el desempeño del screensaver.

- El proyecto ha cumplido con los requisitos indispensables, como el uso de OpenMP, la creación de versiones secuenciales y paralelas del algoritmo, el cálculo del speedup y la implementación del código en C/C++.

- El screensaver desarrollado es estéticamente interesante, con elementos en movimiento y colores generados pseudo aleatoriamente. Se ha incluido un parámetro (N) que indica la cantidad de elementos a renderizar, y se ha incorporado un elemento de física o trigonometría en los cálculos.
- El uso de múltiples funciones de paralelismo en conjunto y la función "parallel for" pueden mejorar significativamente el rendimiento de un programa.
- La elección de la función de paralelismo adecuada dependerá del tipo de tarea que se esté realizando y del hardware en el que se esté ejecutando el programa.
- Es importante medir el tiempo de ejecución de diferentes funciones de paralelismo para determinar cuál es la más adecuada para la tarea que se está realizando.
- La implementación del screensaver ha resultado en una experiencia de usuario agradable, manteniendo los FPS (frames per second) por encima de 30. Esto

demuestra que la paralelización del programa ha sido exitosa en mejorar su rendimiento.

Bibliografia

Chapman, B., Jost, G., Pas, R. V. D., & Kuck, D. J. (2008). Using OpenMP: portable shared memory parallel programming (Vol. 10). Cambridge, MA: MIT press.

Dagum, L., & Menon, R. (1998). OpenMP: An industry-standard API for shared-memory programming. IEEE computational science & engineering, 5(1), 46-55.

Mattson, T. G., Sanders, B. A., & Massingill, B. L. (2004). Patterns for parallel programming. Addison-Wesley Professional.