UNIVERSIDAD DEL VALLE DE GUATEMALA
Computación Paralela y Distribuida, sección 20

Proyecto 1 - Programación Paralela con OpenMP

Sofia Garcia – 22210
Julio Garcia Salas – 22076

GUATEMALA, Septiembre de 2025

Link al Repositorio de Github: https://github.com/Hayser8/CompuParalelaProyecto1

Link al video explicativo: https://youtu.be/1O6ceM-_oyU?si=TaHcCLEG87IXtvpN

Introducción

Este trabajo documenta el diseño y la evaluación de rendimiento de un screensaver con partículas implementado en dos variantes: secuencial y paralela (OpenMP). La aplicación realiza integración por‑frame de la dinámica de partículas y su representación con SDL2, registrando métricas (FPS instantáneo, tiempo de cuadro, *throughput*) para la elaboración de *speedups* y eficiencia. El objetivo es caracterizar la ganancia de paralelización en la etapa de actualización (cómputo CPU) manteniendo el *render* en el hilo principal, y sustentar con datos las conclusiones de desempeño.

Las corridas usadas para este informe —con al menos 10 mediciones por prueba— y los agregados (run_summaries.csv, variant_summary.csv, variant_palette_summary.csv, speedup_by_variant.csv) se encuentran adjuntos en el repositorio de GitHub del proyecto.

Antecedentes

Modelo de paralelización. La variante paralela emplea OpenMP para distribuir el bucle de actualización de partículas entre *threads* mediante parallel for. OpenMP define barreras implícitas al cierre de ciertos constructos y proporciona reducciones para consolidar acumuladores, lo cual permite paralelizar con seguridad la etapa de cómputo independiente por elemento.

Límite teórico de aceleración. La interpretación de resultados se apoya en la ley de Amdahl, que acota el *speedup* alcanzable por la fracción no paralelizable del programa; por tanto, aun con muchos hilos, la sección serial fija un techo de aceleración observable.

Render y presentación. Para la visualización se utiliza SDL2: el dibujo ocurre sobre un *backbuffer* y se "presenta" una vez por cuadro con SDL_RenderPresent; opcionalmente, el *flag* SDL_RENDERER_PRESENTVSYNC sincroniza la presentación con la tasa de refresco del monitor, afectando la cadencia máxima observable de FPS.

Diagrama de flujo del programa

Objetivo. Describir, en orden lógico, el flujo completo del "screensaver" en sus dos variantes (secuencial y paralela), señalando captura de argumentos, ingreso de datos, programación defensiva, secciones paralelas, mecanismos de sincronía y despliegue de resultados.

1) Inicio y captura de argumentos

1. **Inicio.**

2. **Captura.** Se parsean argumentos por CLI: --width, --height, --n, --seconds, --palette={neon|ocean}, --vsync y opciones de logging (ruta/cadencia CSV).

3. **Ayuda.** Si se solicita --help, se imprime uso y se finaliza.

4. **Valores por defecto.** Se establecen por omisión para todo parámetro no provisto.

2) Programación defensiva (validaciones tempranas)

5. **Validación.** Se valida rango/consistencia: width≥640, height≥480, n>0, seconds>0 (si aplica), paleta válida y flags compatibles.

6. **Ruta de error.** Ante error, se registra el motivo, se informa y se aborta antes de reservar recursos.

3) Ingreso de datos e inicialización

7. **Inicialización gráfica.** Se inicializa SDL2; se crean ventana y *renderer*; se cargan texturas (disco/halo) y se configura vsync si procede.

8. **Estado de simulación.** Se inicializa semilla RNG; se reservan/llenan arreglos de Orbiter (partículas) y Attractor.

9. **Relojes y métricas.** Se inicializan contadores de FPS, tiempo por frame (ms) y throughput; se prepara handle de CSV para registro.

4) Bucle principal

10. **Eventos.** Se procesa la cola SDL (teclado/cierre). Si hay salida, se marca condición de fin.

11. **Tiempo.** Se calcula dt (delta) para integración y se actualiza el temporizador global.

4.1) Actualización (dinámica)

12. **Secuencial.** Se itera sobre todas las partículas y se integra su estado (fuerzas/atractores, velocidad, posición, color); se aplican límites/rebotes.

13. **Paralela.** Se paraleliza la actualización con OpenMP (#pragma omp parallel for). Cada hilo integra un subconjunto de partículas.

Mecanismos de sincronía (paralela):

- **Barreras implícitas** al cierre del parallel for.

- **Reducciones** para agregados de métricas cuando corresponde.
- **Separación de lectura/escritura** (doble *buffer* cuando aplica) para evitar *data races*.

4.2) Render y HUD

14. **Dibujo.** Se limpia el *frame*, se dibujan partículas (texturas) y se renderiza HUD (FPS/contadores).
15. **Presentación.** Se presenta el *frame* en pantalla.

4.3) Registro de métricas

16. **Medición por frame.** Se calcula FPS instantáneo y tiempo por frame (ms); se actualizan acumuladores.
17. **CSV.** Según cadencia, se vuelca una fila con: timestamp, FPS, frame_ms, throughput, N, resolución, paleta y flags.

5) Término y salida

18. **Condición de fin.** Si se cumple tiempo objetivo o evento de salida, se abandona el bucle.
19. **Despliegue de resultados.** Se cierran CSV y se imprimen estadísticas del *run* si aplica.
20. **Liberación.** Se destruyen texturas/renderer/ventana y se finaliza SDL2.

Catálogo de funciones

Se listan las funciones y estructuras relevantes identificadas en los archivos screensaver_seq.c (secuencial) y screensaver_paralelo.c (paralela). Formato: Entradas (nombre: tipo – uso), Salidas (retorno y/o *out params*), Descripción (propósito/funcionamiento).

2.1 Funciones (comunes/secuencial)

Función	Entradas	Salidas	Descripción

print_usage(const char *exe)	exe: const char* – nombre del ejecutable	void	Imprime ayuda/sintaxis de uso.
parse_int(const char *s, int *out)	s: const char* – fuente; out: int* – destino	bool + *out	Convierte cadena a entero con validación.
parse_float(const char *s, float *out)	s: const char*; out: float*	bool + *out	Convierte cadena a flotante con validación.
str_ieq(const char *a, const char *b)	a,b: const char*	bool	Compara cadenas sin sensibilidad a mayúsculas.
parse_args(int argc, char **argv)	argc,argv	Config	Parsea CLI, aplica *defaults* y valida.
init_attractors(Attractor a[NUM_ATTR], int W, int H)	a: Attractor[], W/H: int	void (+ a por referencia)	Inicializa posición/estado de atractores.
update_attractors(Attractor a[NUM_ATTR], float t, int W, int H)	a: Attractor[], t: float, W/H: int	void (+ a)	Actualiza atractores en función del tiempo.
init_orbiters(Orbiter *o, int n, Attractor a[NUM_ATTR], int W, int H)	o: Orbiter*, n: int, a: Attractor[], W/H: int	void (+ o)	Inicializa partículas con condiciones iniciales.
update_orbiters(Orbiter *o, int n, Attractor a[NUM_ATTR], float dt, int W, int H)	o, n, a, dt, W/H	void (+ o)	Integra dinámica de partículas (secuencial).

palette_attractor_color(const char *pal, int k, float t, Uint8 *r, *g, *b)	pal: const char*, k: int, t: float	void + *r,*g,*b	Devuelve color de atractor según paleta/tiempo.
palette_color(const Config *cfg, int i, float t, Uint8 *r, *g, *b)	cfg, i: int, t: float	void + *r,*g,*b	Color por partícula (según paleta/config.).
render_frame(SDL_Renderer *ren, Orbiter *o, int n, Attractor a[NUM_ATTR], int W, int H, float t, const Config *cfg)	ren, o, n, a, W/H, t, cfg	void	Dibuja partículas y HUD; presenta *frame*.
main(int argc, char **argv)	argc,argv	int	Orquestación del ciclo de vida del programa.

2.2 Funciones (paralela)

Función	Entradas	Salidas	Descripción
update_orbiters_parallel(Orbiter *o, int n, Attractor a[NUM_ATTR], float dt, int threads)	o,n,a,dt,threads	void (+ o)	Integra partículas en paralelo con OpenMP (#pragma omp parallel for). Barrera implícita al cierre del bucle.
precalc_particles(const Config *cfg, const Orbiter *o, int n, float t, float cx, float cy, Precomp *out, int threads)	cfg,o,n,t,cx,cy,threads	void + *out	Precalcula datos intermedios de dibujo en Precomp para separar cómputo de *render*.

particle_color(const Config *cfg, int i, float t, Uint8 *r, *g, *b)	cfg,i,t	void + *r,*g,*b	Color por partícula (versión usada en paralelo).
render_frame(SDL_Renderer *ren, const Config *cfg, const Precomp *pc, …, SDL_Texture **discs, SDL_Texture *radial)	ren,cfg,pc,…	void	Dibujo final y HUD (usa *precomputación* y texturas).

Además de las **comunes** listadas en 2.1 (presentes también en la versión paralela).

2.3 Estructuras (tipos) relevantes

Tipo	Descripción (uso)
Config	Parámetros de ejecución (resolución, N, duración, paleta, vsync, logging, etc.).
Attractor	Estado de cada atractor (posición y parámetros temporales).
Orbiter	Estado de cada partícula (posición, velocidad, color/índice).
FPSCounter	Acumuladores y ventana para cálculo de FPS y *frame time*.
RGBA	Color en 8 bits por canal.
Precomp *(paralela)*	Contenedor de datos intermedios precalculados para acelerar el *render*.

Bitácora de pruebas

Diseño general: Se ejecutaron corridas independientes para ambas variantes (secuencial y paralela), con al menos 10 mediciones por prueba y registro por‑frame a CSV (FPS instantáneo, frame_ms, throughput, configuración). Para trazabilidad, en el repositorio de GitHub se adjuntan los datos fuente

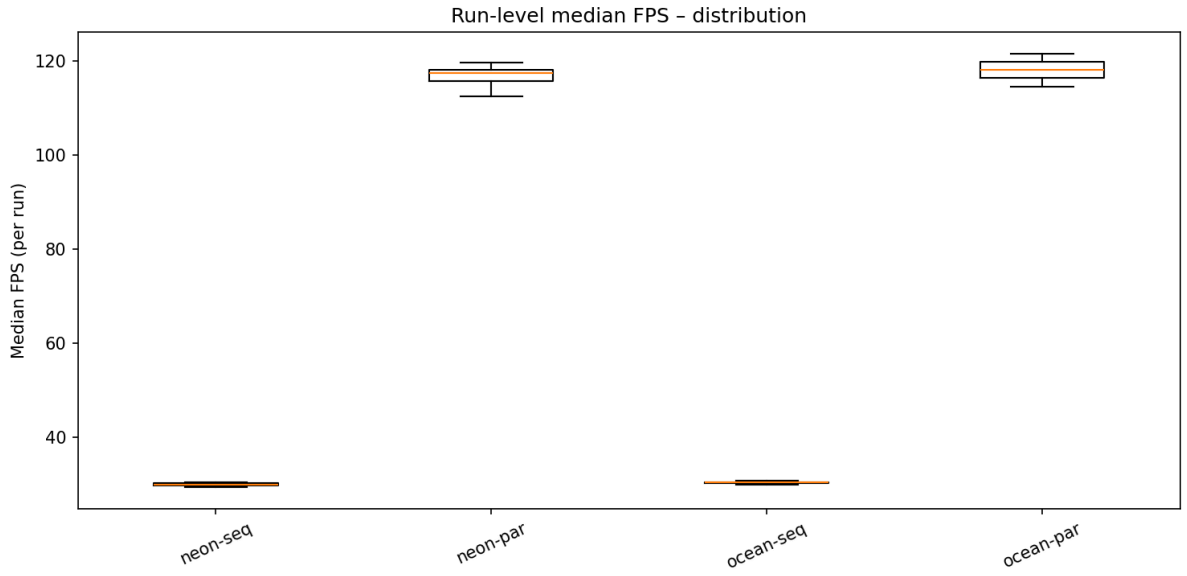
por corrida y agregados: run_summaries.csv, variant_summary.csv, variant_palette_summary.csv, speedup_by_variant.csv, además de los archivos por run (par_neon_*.csv, par_ocean_*.csv, etc.).

3.1 Resultados globales (resumen numérico)

- **FPS (mediana):** Secuencial ≈ 30.23 vs Paralelo ≈ 117.57 → speedup ≈ 3.89×.
- **Jitter (p95 frame time, ms):** Secuencial ≈ 34.73 ms vs Paralelo ≈ 16.86 ms → ~2.06× menor.
- **Throughput (mediana, partículas/s):** Secuencial ≈ 42.3k vs Paralelo ≈ 117.6k.
- **Eficiencia** (con 10 hilos reportados): ≈ 38.9% (speedup/hilos).

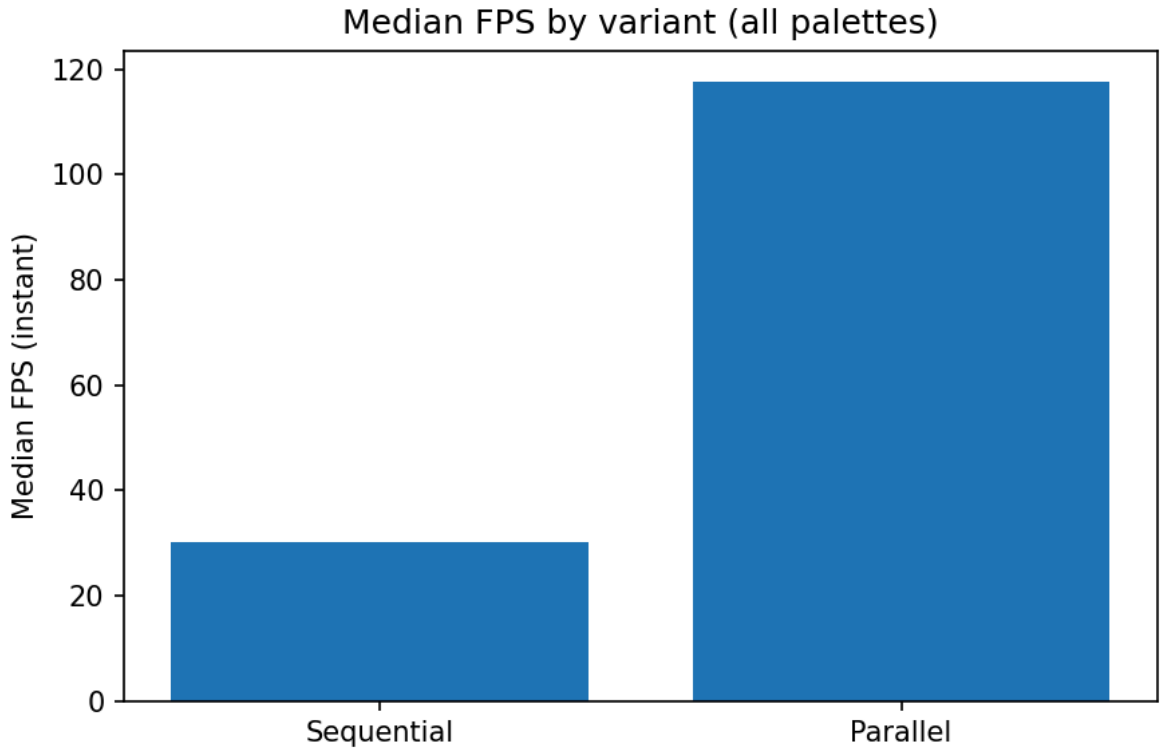
3.2 Figuras y comentarios

Figura 1 — Distribución del FPS mediano por corrida



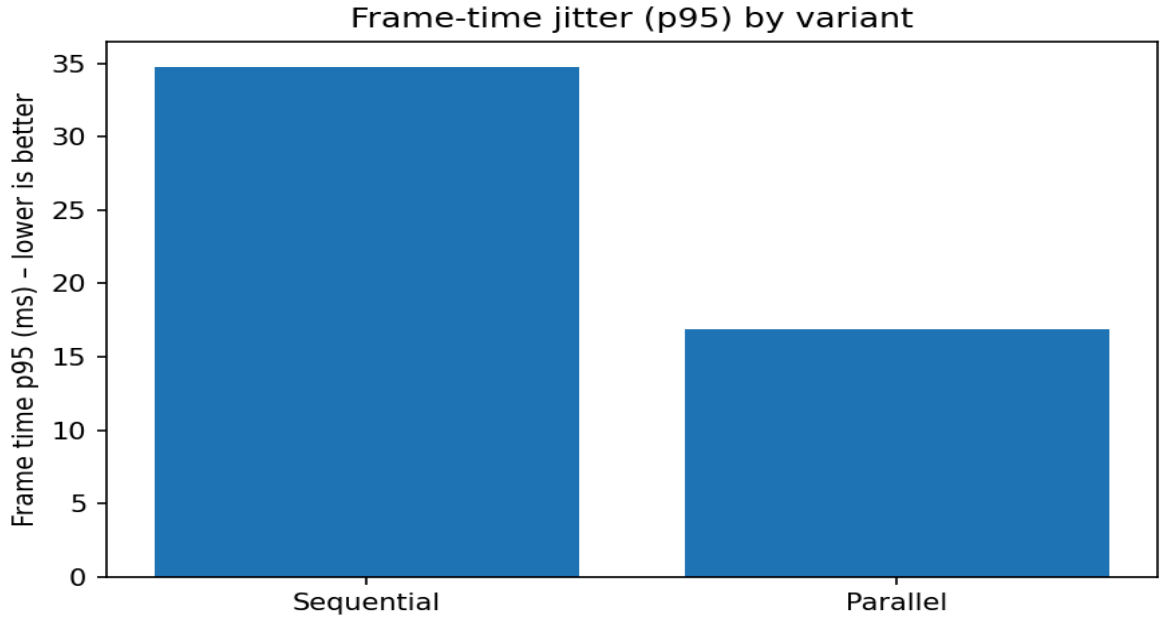
La variante paralela (neon-par y ocean-par) muestra medianas agrupadas en ~116–120 FPS con dispersión acotada; la secuencial (neon-seq y ocean-seq) se concentra alrededor de ~30 FPS. La separación entre cajas confirma una ganancia consistente en todas las corridas, no solo en valores puntuales.

Figura 2 — FPS mediano por variante (todas las paletas)



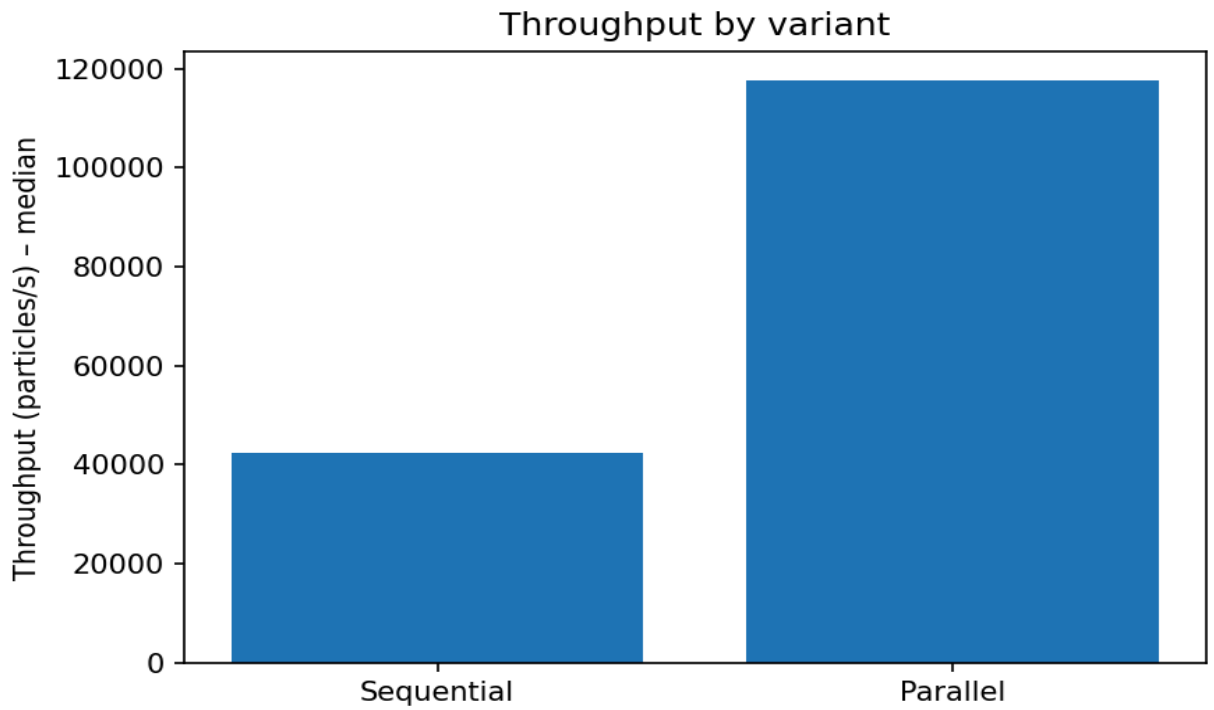
La barra Paralelo supera ampliamente a Secuencial (≈117.6 vs 30.2 FPS). Este contraste sintetiza el resultado principal: mejora de ~3.89× en rendimiento percibido.

Figura 3 — Jitter (p95 de tiempo de frame, ms) por variante



El p95 del tiempo de frame desciende de ~34.7 ms (Secuencial) a ~16.9 ms (Paralelo). Un p95 menor implica mayor estabilidad temporal: menos picos de latencia y una animación más uniforme (cercana/superior a 60 FPS sostenidos).

Figura 4 — Throughput (partículas/s, mediana) por variante



El throughput mediano aumenta de ~42.3k (Secuencial) a ~117.6k (Paralelo). Aun considerando diferencias de N y resolución entre variantes, el orden de magnitud se mantiene y evidencia mayor trabajo útil por segundo en la versión paralela.

3.3 Evidencia y archivos adjuntos

- **Datos por corrida y agregados**: disponibles en el repositorio de GitHub (CSV indicados arriba), junto con las figuras referidas en este anexo.

- **Réplica de resultados**: cada CSV incluye configuración (resolución, N, paleta, hilos, vsync) y series temporales para reproducir gráficos, speedups y eficiencia.

Conclusiones / Recomendaciones

Conclusiones.

- Con la configuración de medición utilizada, la variante paralela mostró una mejora de ~3.89× en FPS mediano frente a la secuencial (≈117.6 vs ≈30.2 FPS), con reducción del jitter p95 del tiempo de cuadro de ~34.7 ms a ~16.9 ms y un incremento de *throughput* (partículas/s) de ~42.3 k a ~117.6 k.

- La consistencia entre paletas (neon y ocean) confirma que el beneficio proviene del esquema de paralelización y no de la estética elegida.

- Dado el modelado de OpenMP y la ley de Amdahl, parte del *frame* permanece serial (render/HUD y porciones de la lógica), lo que explica que el *speedup* práctico sea sustancial pero inferior al número de hilos.

Recomendaciones (para el informe y la lectura de resultados).

- Incluir en la sección de metodología que, en este conjunto de corridas, resolución y N difieren entre variantes; los CSV del repositorio permiten filtrar corridas con la misma configuración si se desea aislar el efecto puro de paralelización.

- Mantener en anexos las figuras comparativas (FPS mediano, p95 de *frame time*, *throughput*) y vincular cada gráfica con el CSV de origen para trazabilidad.

- Al discutir escalabilidad, referenciar brevemente Amdahl y el modelo de barreras/reducciones de OpenMP para contextualizar por qué el *speedup* observado es coherente con la fracción no paralelizable.

Referencias

How to draw circles, arcs and vector graphics in SDL? (s/f-b). Stack Overflow. Recuperado el 4 de septiembre de 2025, de https://stackoverflow.com/questions/38334081/how-to-draw-circles-arcs-and-vector-graphics-in-sdl

SDL library in C/C++ with examples. (2019, marzo 26). GeeksforGeeks. https://www.geeksforgeeks.org/c/sdl-library-in-c-c-with-examples/

SDL2_gfx: File List. (s/f). Ferzkopp.net. Recuperado el 4 de septiembre de 2025, de https://www.ferzkopp.net/Software/SDL2_gfx/Docs/html/files.html