



UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Ecuación de Poisson 2D

Paralelización con Open-MP

*Laura Oliveros
Andrés Gómez
Julián Aros*

*Programa Académico de Física
Universidad Distrital Francisco José de Caldas*

Capacidad de la máquina

```
admin@ip-172-31-21-223:~/Taller_OpenMP_Poisson$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Address sizes:         46 bits physical, 48 bits virtual
Byte Order:            Little Endian
CPU(s):                32
On-line CPU(s) list:   0-31
Vendor ID:             GenuineIntel
Model name:            Intel(R) Xeon(R) Platinum 8375C CPU @ 2.90GHz
CPU family:            6
Model:                 106
Thread(s) per core:    2
Core(s) per socket:    16
Socket(s):             1
Stepping:              6
BogoMIPS:              5800.03
```

Ejemplos Tratados: Condiciones de Frontera

Ejemplo 0: Término Fuente Gaussiano

Ecuación Diferencial

$$\nabla^2 V = f(x, y) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x - \mu)^2 + (y - \mu)^2}{2\sigma^2}\right)$$

Dominio

donde $\mu = 0,5$ y $\sigma = 0,1$.

$$\mathcal{D} = [0, 1] \times [0, 1]$$

Condiciones de Frontera

$$V(0, y) = 0 \quad \text{para } y \in [0, 1] \text{ (frontera izquierda)}$$

$$V(1, y) = 0 \quad \text{para } y \in [0, 1] \text{ (frontera derecha)}$$

$$V(x, 0) = 0 \quad \text{para } x \in [0, 1] \text{ (frontera inferior)}$$

$$V(x, 1) = x \quad \text{para } x \in [0, 1] \text{ (frontera superior)}$$

Ejemplo 1: Término Fuente Exponencial

Ecuación Diferencial

$$\nabla^2 V = (x^2 + y^2)e^{xy}$$

Dominio

$$\mathcal{D} = [0, 2] \times [0, 1]$$

Condiciones de Frontera

Solución Analítica

$$V(0, y) = 1 \quad \text{para } y \in [0, 1]$$

$$V(2, y) = e^{2y} \quad \text{para } y \in [0, 1]$$

$$V(x, 0) = 1 \quad \text{para } x \in [0, 2]$$

$$V(x, 1) = e^x \quad \text{para } x \in [0, 2]$$

$$V(x, y) = e^{xy}$$

Ejemplo 2: Ecuación de Laplace

Ecuación Diferencial

$$\nabla^2 V = 0$$

Dominio

$$\mathcal{D} = [1, 2] \times [0, 1]$$

Condiciones de Frontera

Solución Analítica

$$V(x, y) = \ln(x^2 + y^2)$$

$$V(1, y) = \ln(y^2 + 1) \quad \text{para } y \in [0, 1]$$

$$V(2, y) = \ln(y^2 + 4) \quad \text{para } y \in [0, 1]$$

$$V(x, 0) = 2 \ln(x) \quad \text{para } x \in [1, 2]$$

$$V(x, 1) = \ln(x^2 + 1) \quad \text{para } x \in [1, 2]$$

Ejemplo 3: Término Fuente Constante

Ecuación Diferencial

$$\nabla^2 V = 4$$

Dominio

$$\mathcal{D} = [1, 2] \times [0, 2]$$

Condiciones de Frontera

$$V(1, y) = (1 - y)^2 \quad \text{para } y \in [0, 2]$$

$$V(2, y) = (2 - y)^2 \quad \text{para } y \in [0, 2]$$

$$V(x, 0) = x^2 \quad \text{para } x \in [1, 2]$$

$$V(x, 2) = (x - 2)^2 \quad \text{para } x \in [1, 2]$$

Solución Analítica

$$V(x, y) = (x - y)^2$$

Ejemplo 4: Término Fuente Racional

Ecuación Diferencial

$$\nabla^2 V = \frac{x}{y} + \frac{y}{x}$$

Dominio

$$\mathcal{D} = [1, 2] \times [1, 2]$$

Condiciones de Frontera

$$V(1, y) = y \ln(y) \quad \text{para } y \in [1, 2]$$

$$V(2, y) = 2y \ln(2y) \quad \text{para } y \in [1, 2]$$

$$V(x, 1) = x \ln(x) \quad \text{para } x \in [1, 2]$$

$$V(x, 2) = x \ln(4x^2) \quad \text{para } x \in [1, 2]$$

Solución Analítica

$$V(x, y) = xy \ln(xy)$$

Preparación

Ejemplo 0: Código general sin modificación (Original)

Tiempo de ejecución: 0,588 segundos
Número de iteraciones: 7609 iteraciones
Tolerancia alcanzada: 9.53674e-07
Número de threads: 1

Ejemplo 1:

Tiempo de ejecución: 0,186 segundos
Número de iteraciones: 2432 iteraciones
Tolerancia alcanzada: 9.97117e-07
Número de threads: 1

Ejemplo 2:

Tiempo de ejecución: 0,168 segundos
Número de iteraciones: 2204 iteraciones
Tolerancia alcanzada: 9.98661e-07
Número de threads: 1

Ejemplo 3:

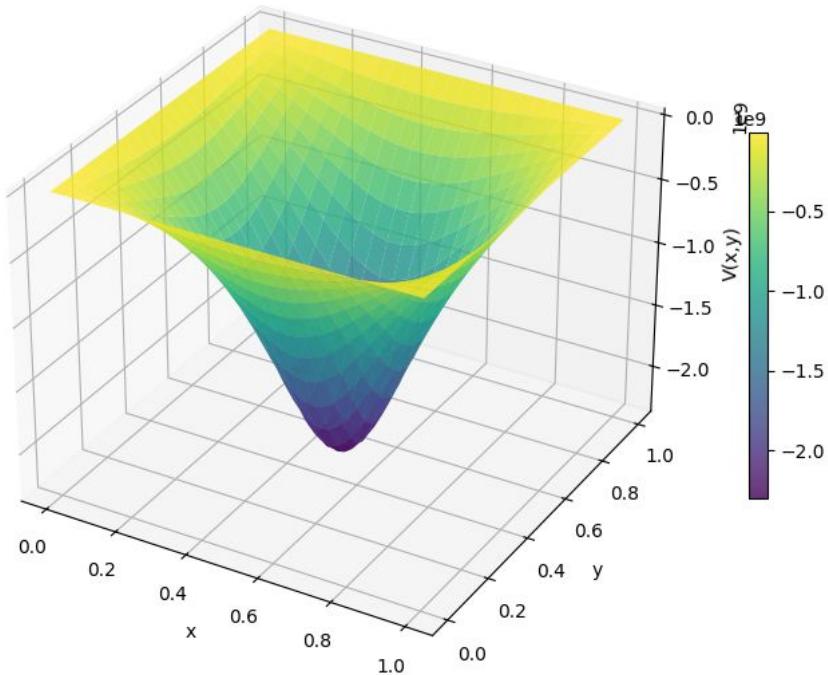
Tiempo de ejecución: 0,156 segundos
Número de iteraciones: 2040 iteraciones
Tolerancia alcanzada: 9.9625e-07
Número de threads: 1

Ejemplo 4:

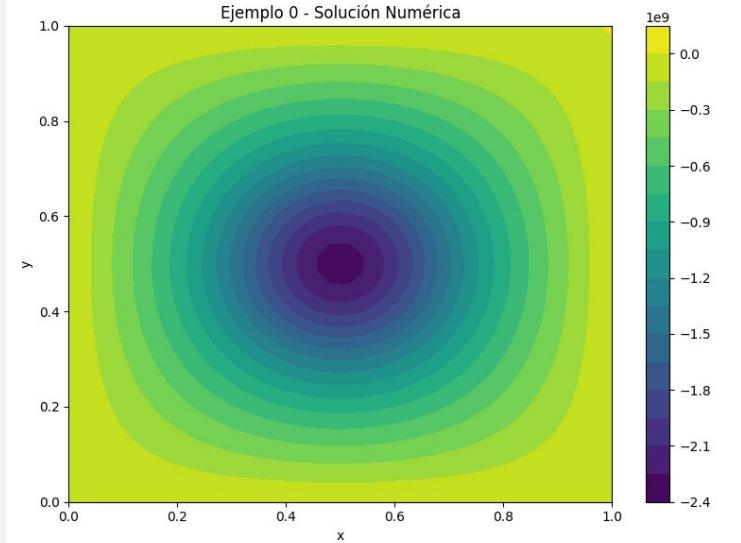
Tiempo de ejecución: 0,184 segundos
Número de iteraciones: 2415 iteraciones
Tolerancia alcanzada: 9.96643e-07
Número de threads: 1

Resultados Serial: Original

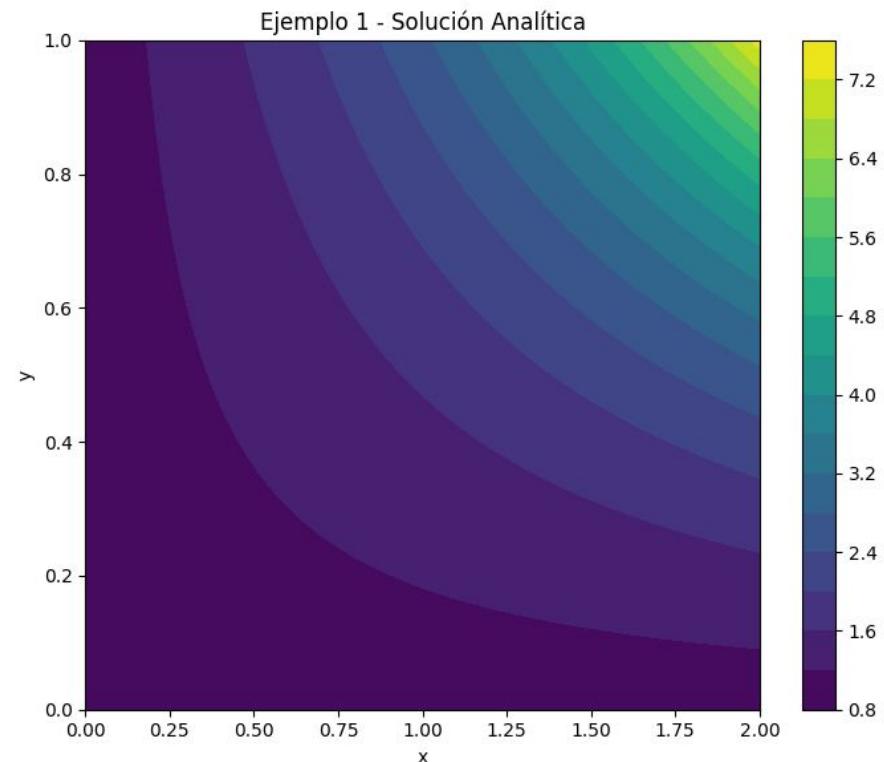
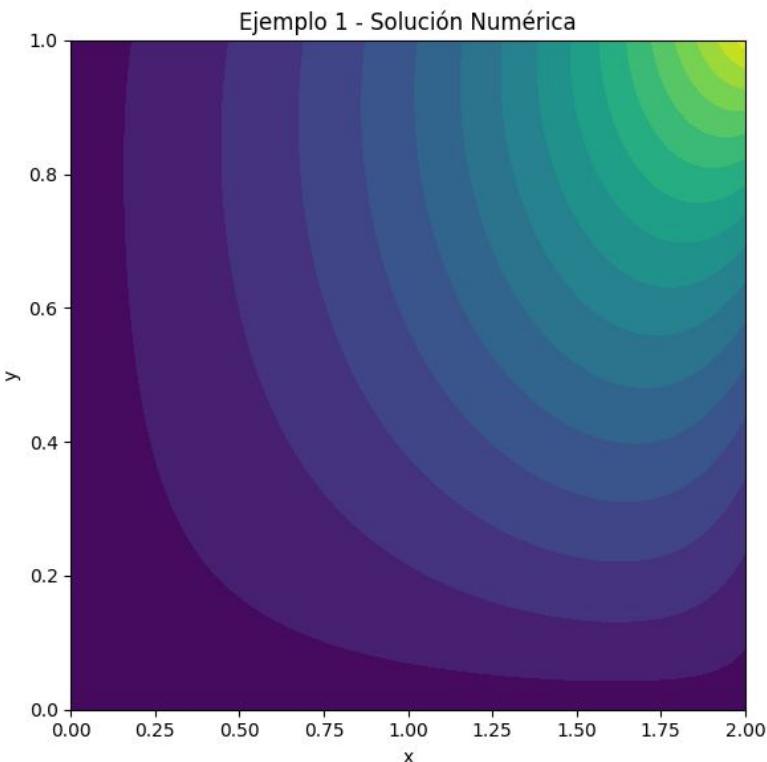
Ejemplo 0 - Solución Numérica (3D)



Ejemplo 0 - Solución Numérica

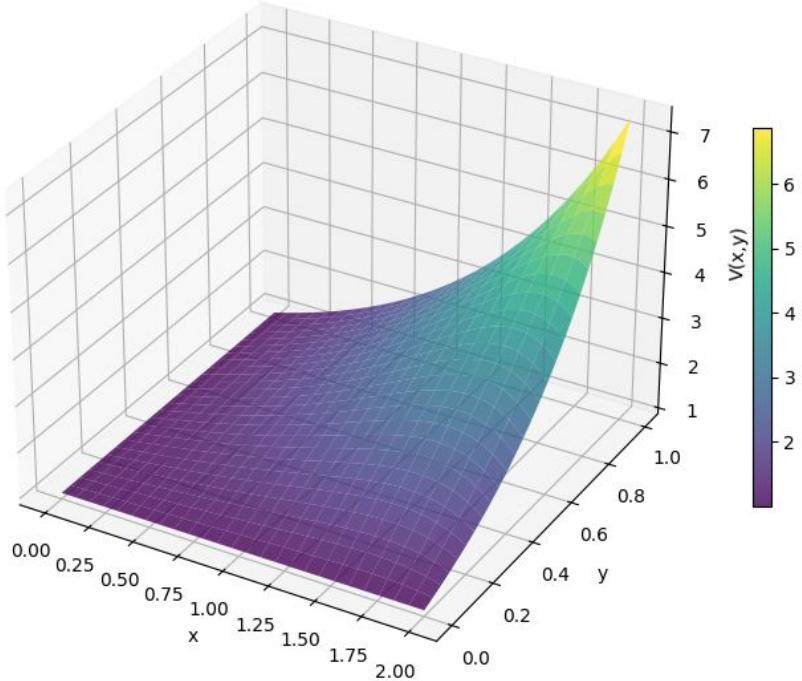


Resultados Serial: ejemplo 1

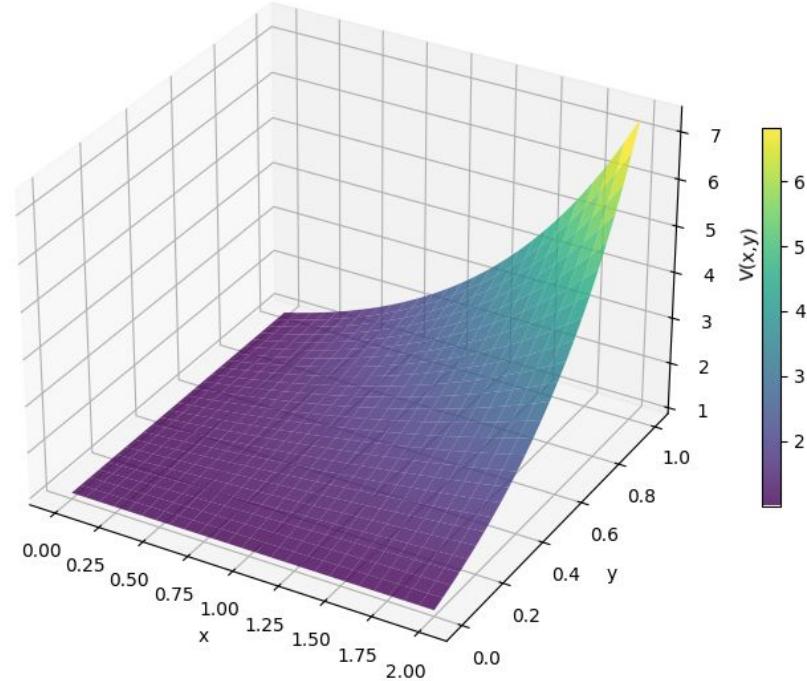


Resultados Serial: ejemplo 1

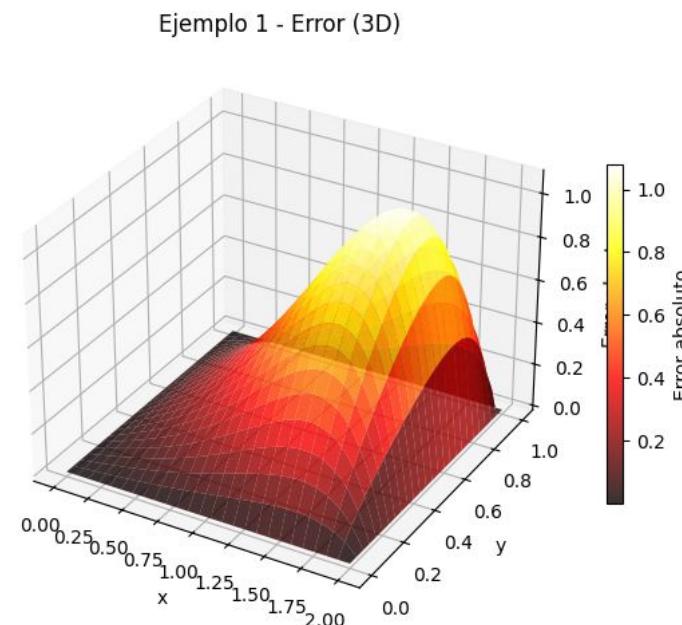
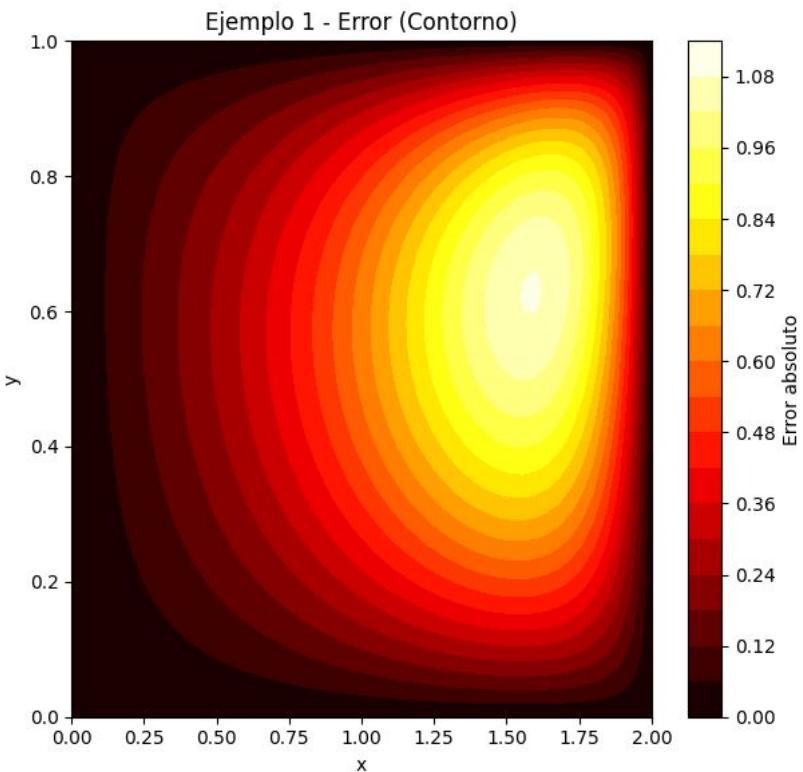
Ejemplo 1 - Solución Numérica (3D)



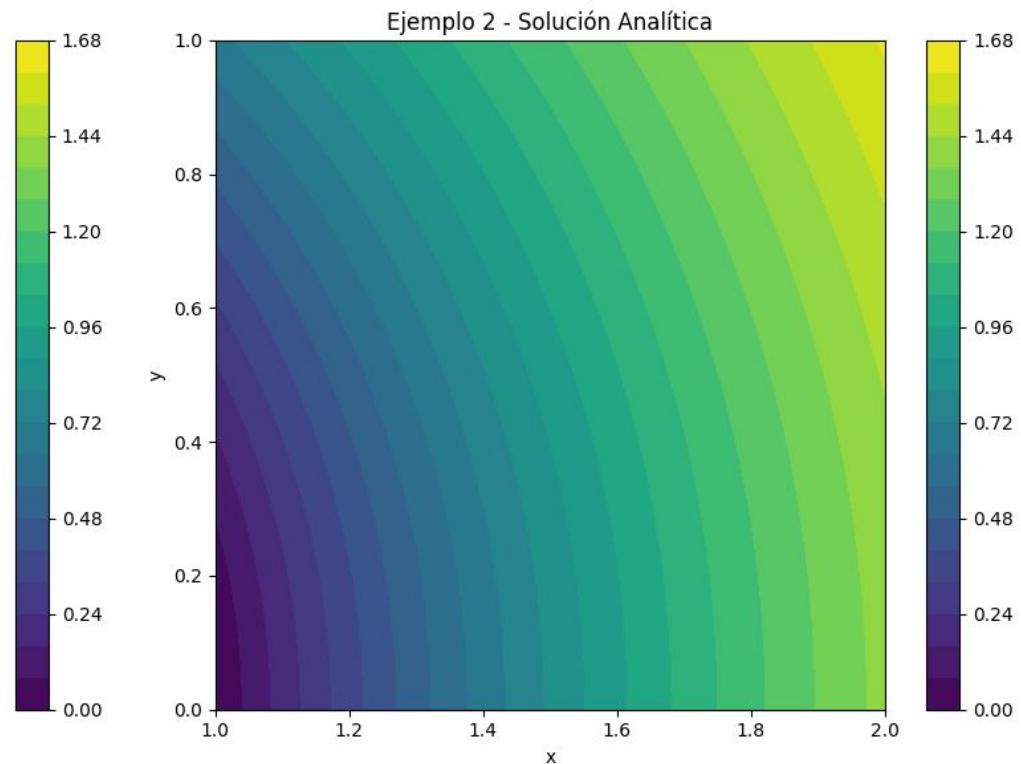
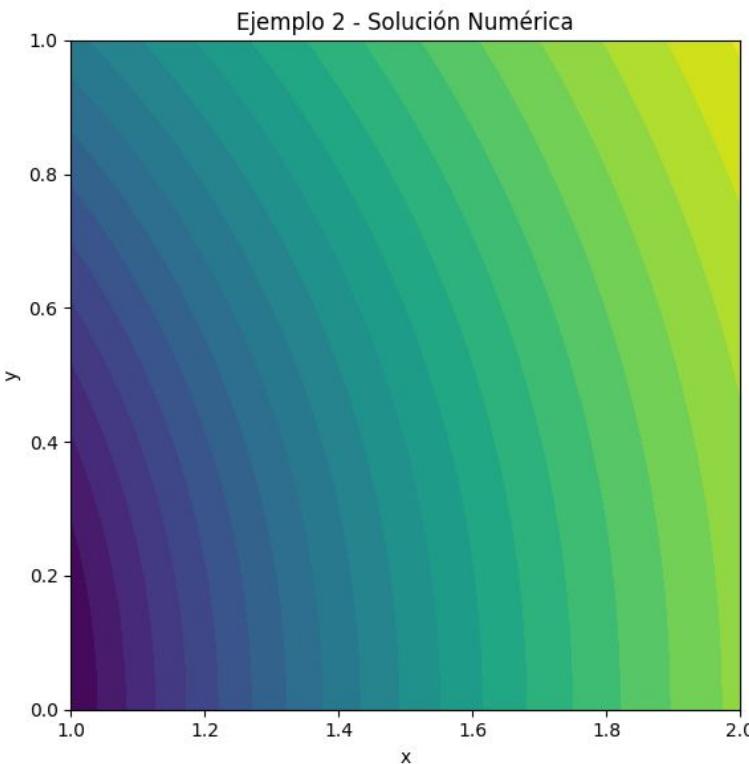
Ejemplo 1 - Solución Analítica (3D)



Resultados Serial: ejemplo 1

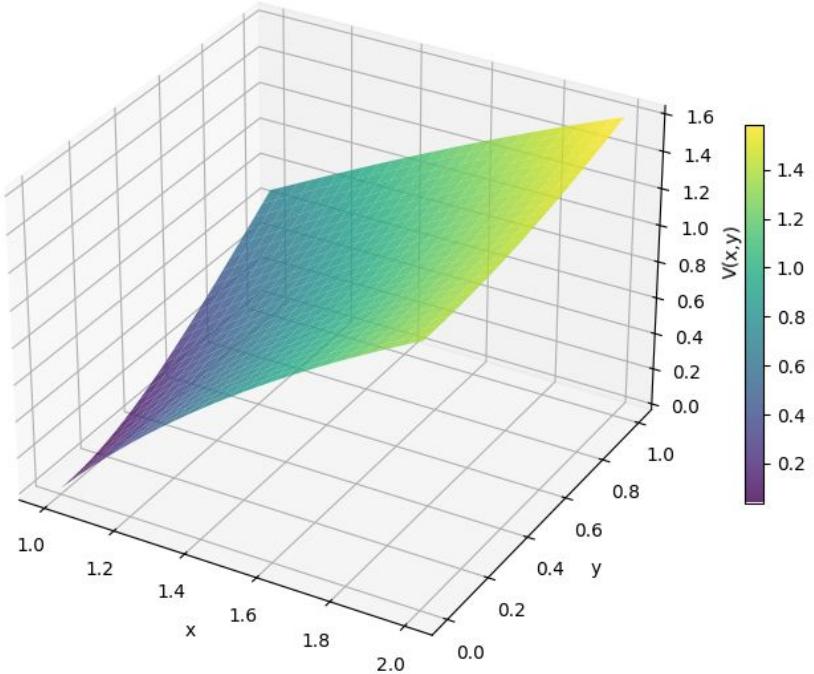


Resultados Serial: ejemplo 2

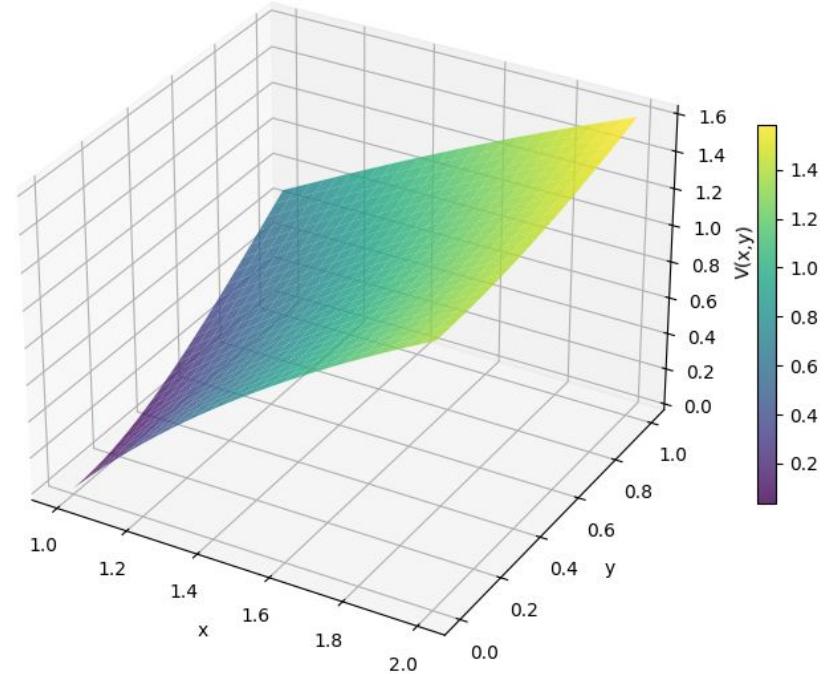


Resultados Serial: ejemplo 2

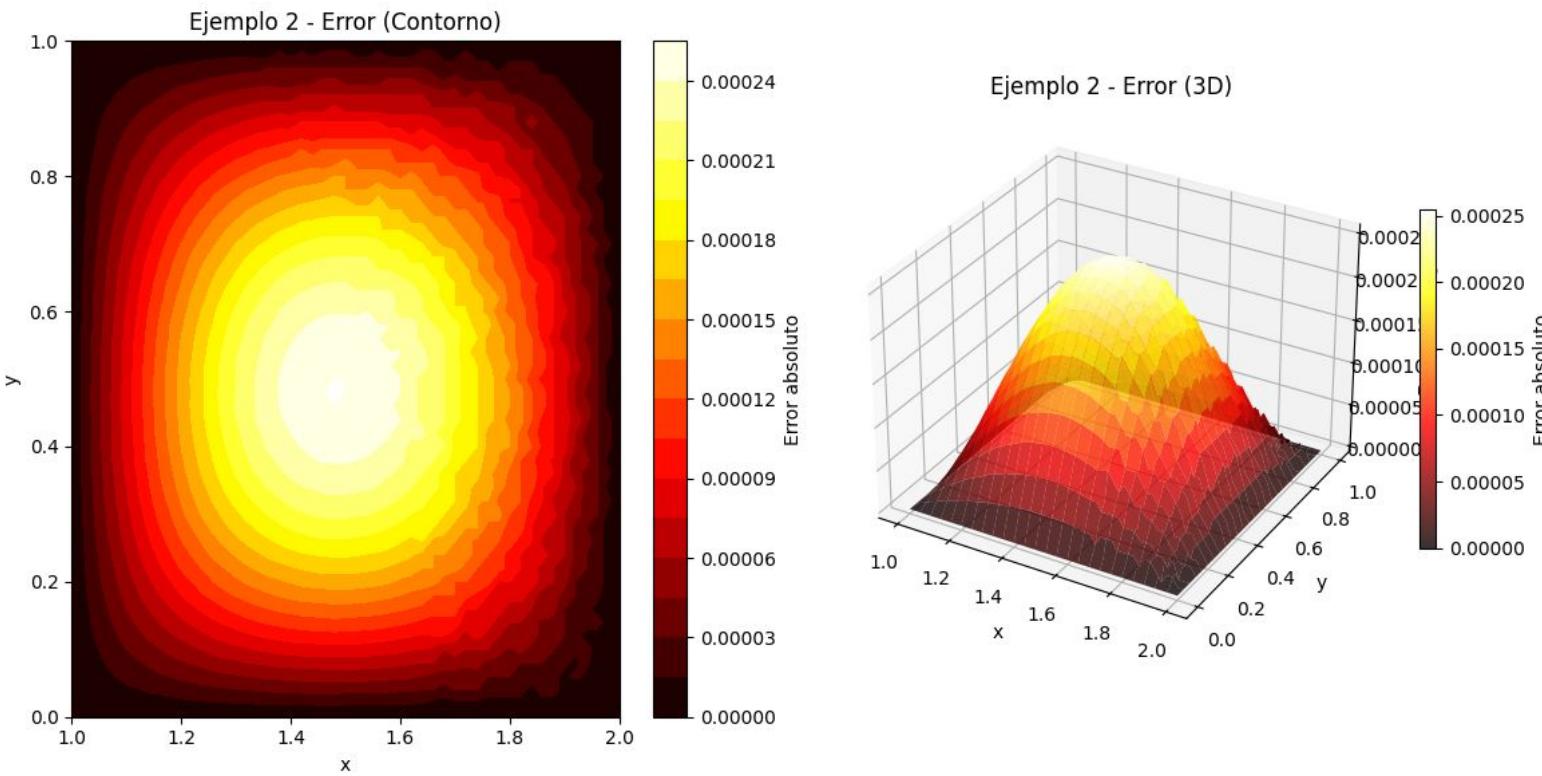
Ejemplo 2 - Solución Numérica (3D)



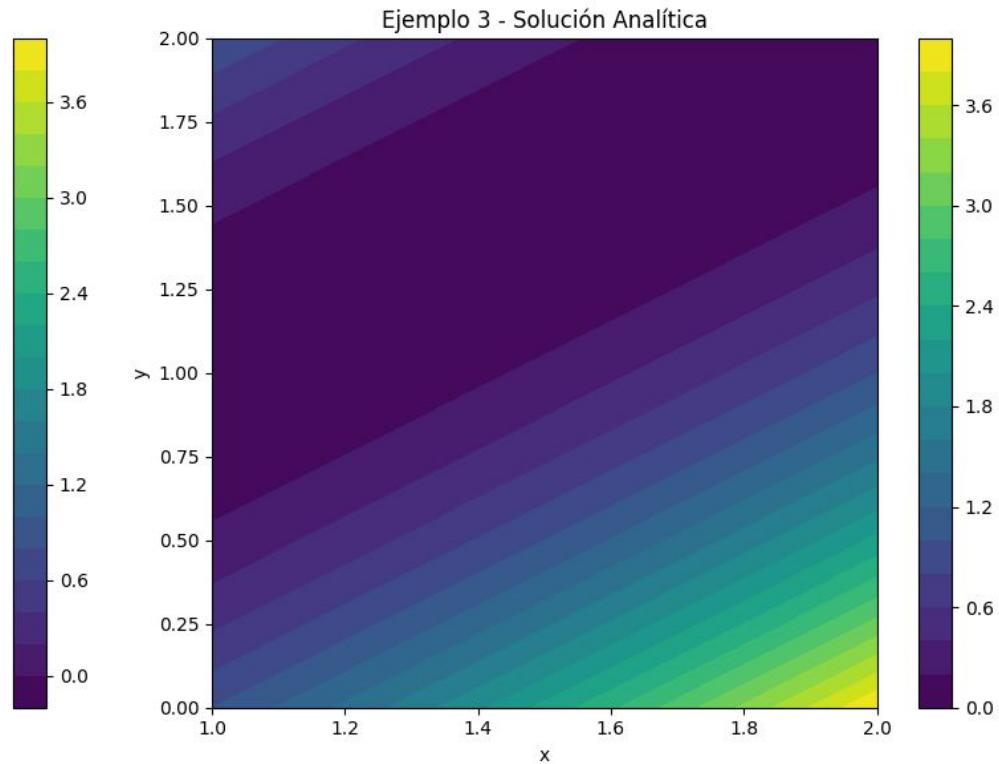
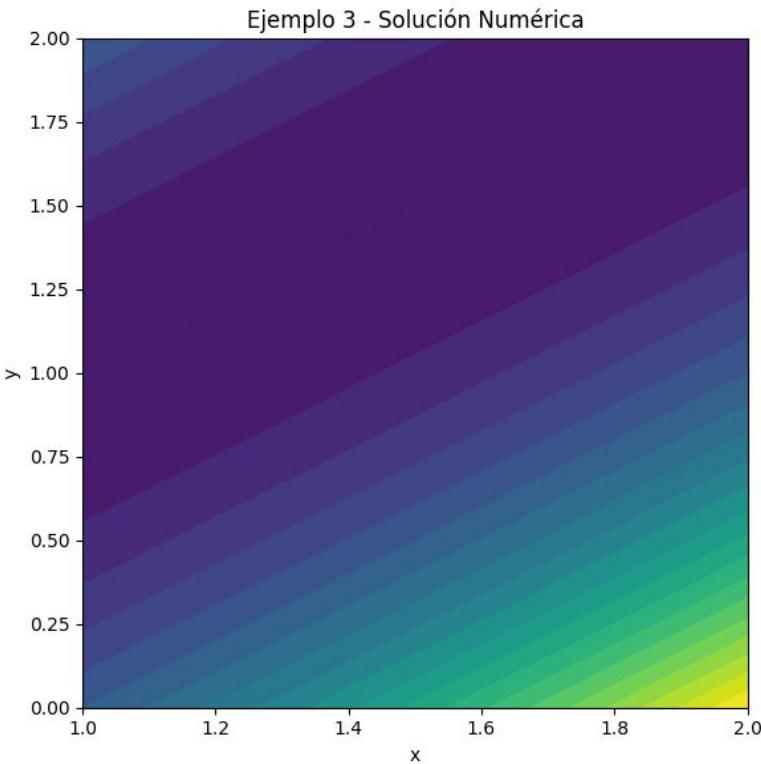
Ejemplo 2 - Solución Analítica (3D)



Resultados Serial: ejemplo 2

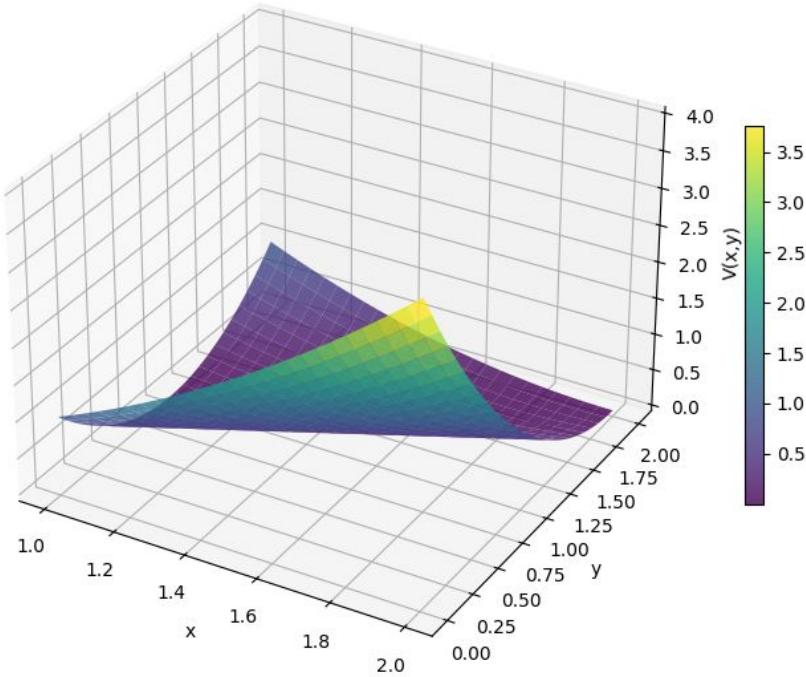


Resultados Serial: ejemplo 3

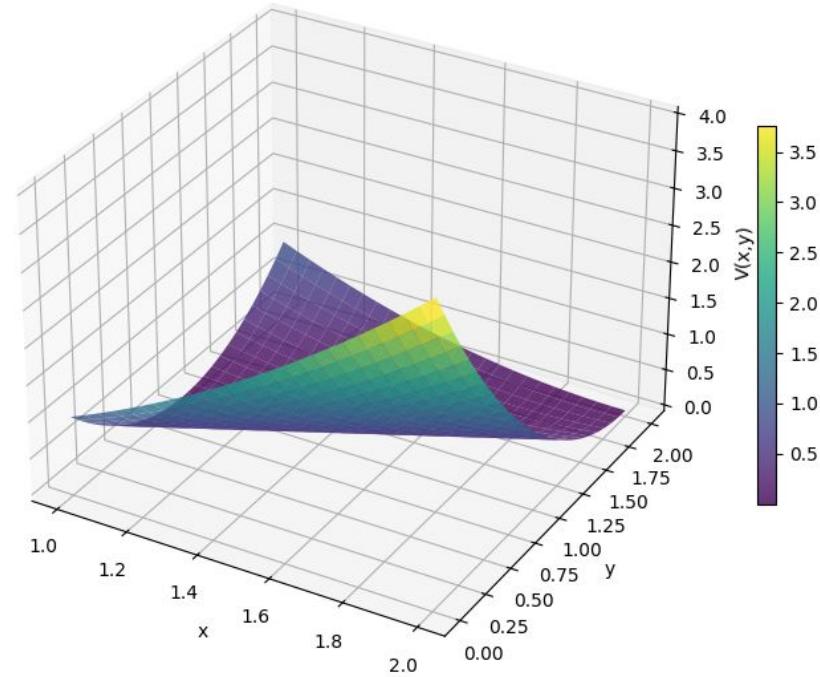


Resultados Serial: ejemplo 3

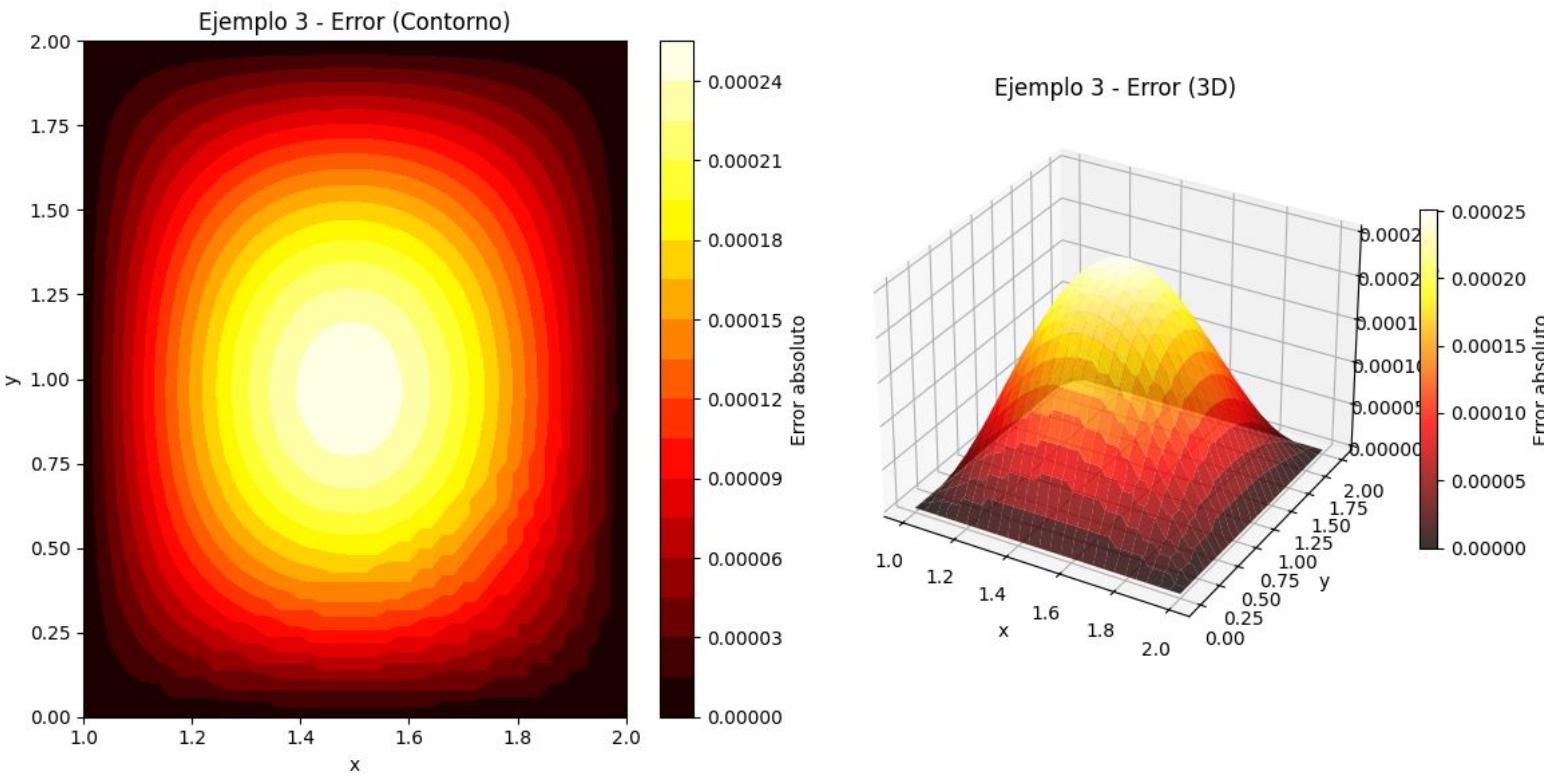
Ejemplo 3 - Solución Numérica (3D)



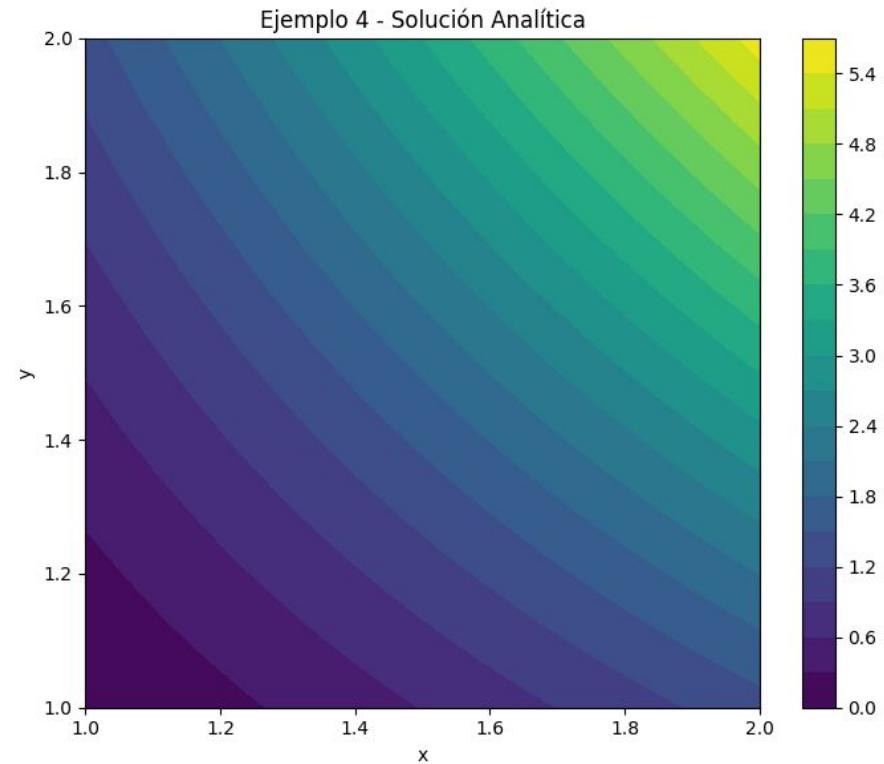
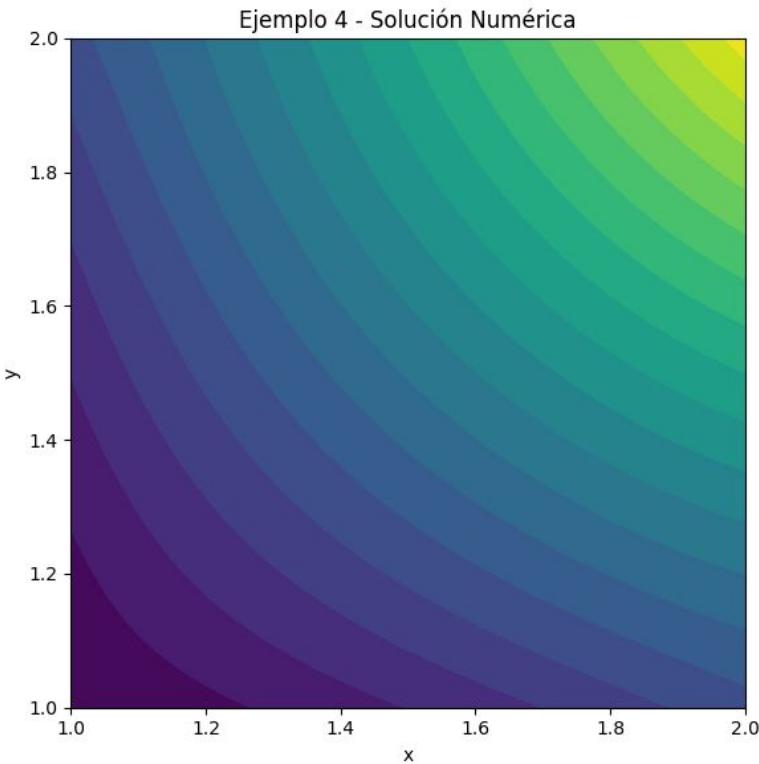
Ejemplo 3 - Solución Analítica (3D)



Resultados Serial: ejemplo 3

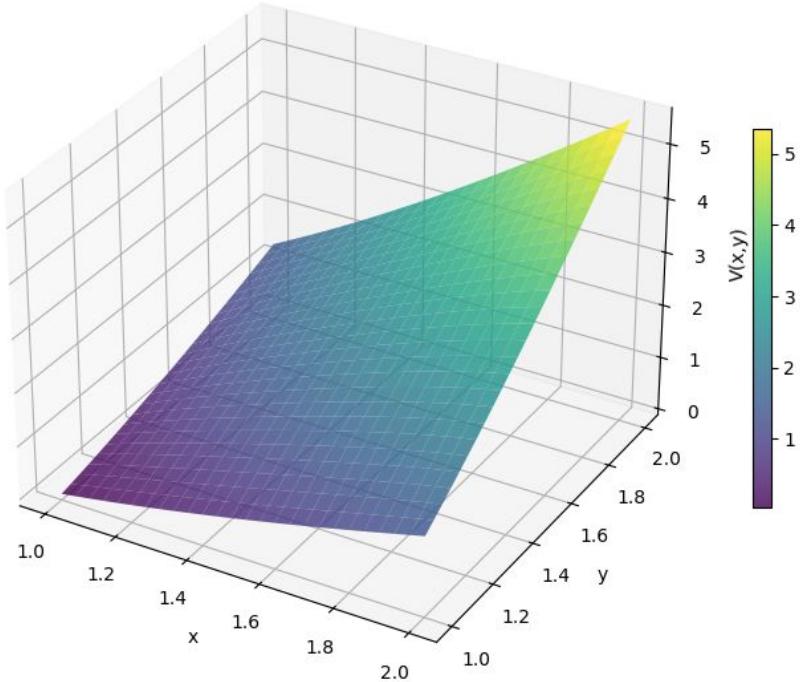


Resultados Serial: ejemplo 4

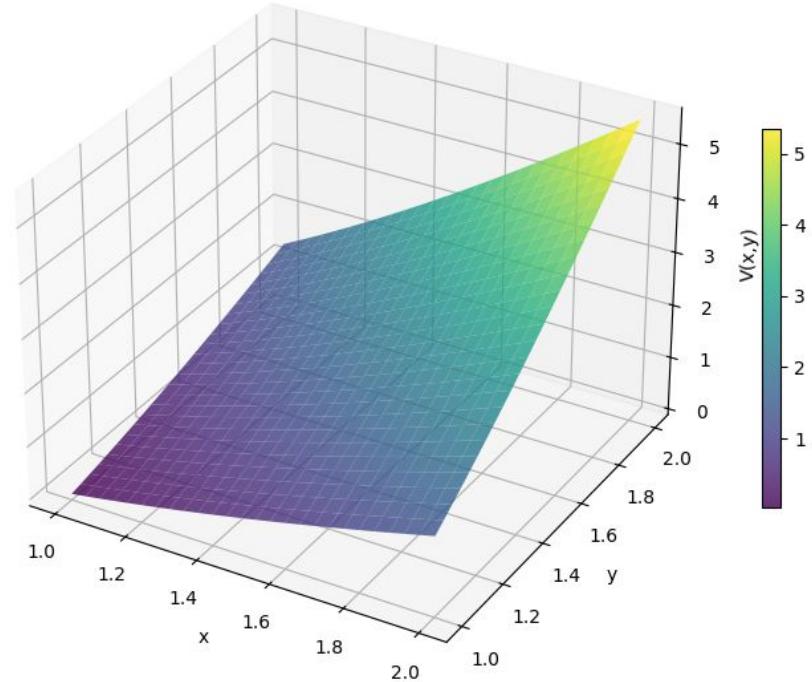


Resultados Serial: ejemplo 4

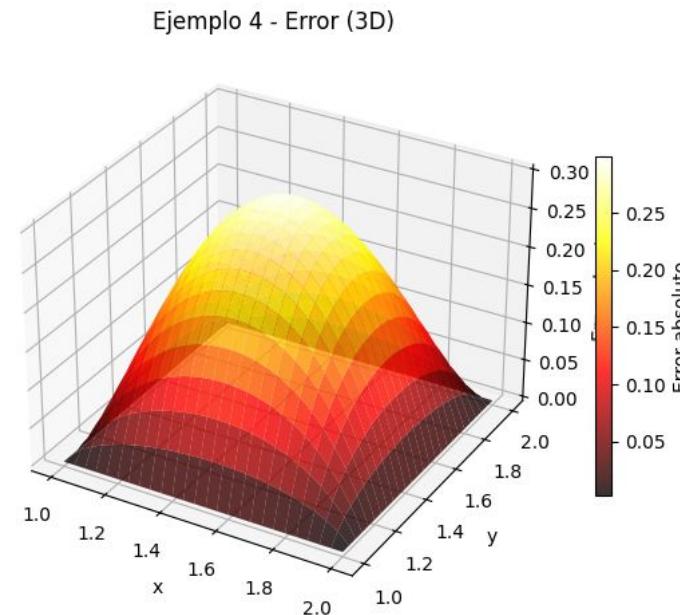
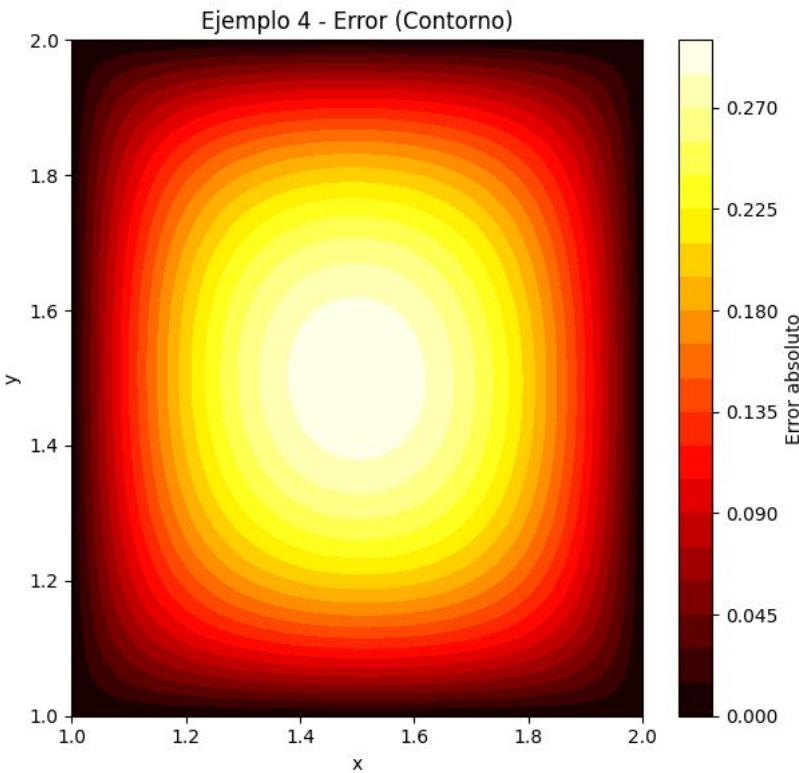
Ejemplo 4 - Solución Numérica (3D)



Ejemplo 4 - Solución Analítica (3D)



Resultados Serial: ejemplo 4



Actividad 0: Código serial malla 300x300

- Paralelizar el bucle principal en `solve_poisson`.

```
admin@ip-172-31-21-223:~/Taller_OpenMP_Poisson/src$ ./actividad_1
== ACTIVIDAD 1: SIMULADOR DE ECUACIÓN DE POISSON 2D (#pragma omp parallel for) ==
Directorio 'actividad1' creado exitosamente.
```

Ejemplos disponibles:

- 0 - Ejemplo Original: Término fuente gaussiano
- 1 - Ejemplo 1: $\nabla^2 V = (x^2 + y^2)e^{xy}$
- 2 - Ejemplo 2: $\nabla^2 V = 0$ (Ecuación de Laplace)
- 3 - Ejemplo 3: $\nabla^2 V = 4$
- 4 - Ejemplo 4: $\nabla^2 V = x/y + y/x$
- 5 - Ejecutar todos los ejemplos

Seleccione una opción (0-5): 0

Resultados: Actividad 0

Ejemplo 0

```
==> RESULTADOS ==>
Tiempo de ejecución: 693.216 segundos
Número de iteraciones: 246519
Tolerancia alcanzada: 9.53674e-07
Número de threads: 1 (secuencial)
```

Ejemplo 1

```
==> RESULTADOS ==>
Tiempo de ejecución: 152.419 segundos
Número de iteraciones: 54646
Tolerancia alcanzada: 9.99952e-07
Número de threads: 1 (secuencial)
```

Ejemplo 2

```
==> RESULTADOS ==>
Tiempo de ejecución: 129.502 segundos
Número de iteraciones: 46484
Tolerancia alcanzada: 9.99925e-07
Número de threads: 1 (secuencial)
```

Ejemplo 3

```
==> RESULTADOS ==>
Tiempo de ejecución: 113.963 segundos
Número de iteraciones: 40895
Tolerancia alcanzada: 9.99933e-07
Número de threads: 1 (secuencial)
```

Ejemplo 4

```
==> RESULTADOS ==>
Tiempo de ejecución: 150.358 segundos
Número de iteraciones: 53975
Tolerancia alcanzada: 9.99965e-07
Número de threads: 1 (secuencial)
```

Resultados: Actividad 0

Ejemplo	Tiempo de ejecución (s)	Número de iteraciones
0	693.216	246519
1	152.419	54646
2	129.502	46484
3	113.963	40895
4	150.358	53975

Actividades de paralelización

Actividad 1: #pragma omp parallel for

- Paralelizar el bucle principal en `solve_poisson`.

```
admin@ip-172-31-21-223:~/Taller_OpenMP_Poisson/src$ ./actividad_1
== ACTIVIDAD 1: SIMULADOR DE ECUACIÓN DE POISSON 2D (#pragma omp parallel for) ==
Directorio 'actividad1' creado exitosamente.
```

Ejemplos disponibles:

- 0 - Ejemplo Original: Término fuente gaussiano
- 1 - Ejemplo 1: $\nabla^2 V = (x^2 + y^2)e^{xy}$
- 2 - Ejemplo 2: $\nabla^2 V = 0$ (Ecuación de Laplace)
- 3 - Ejemplo 3: $\nabla^2 V = 4$
- 4 - Ejemplo 4: $\nabla^2 V = x/y + y/x$
- 5 - Ejecutar todos los ejemplos

Seleccione una opción (0-5): 0

Resultados: Actividad 1

Ejemplo 0

```
== EJEMPLO 0: Término fuente gaussiano ==
Dominio: x ∈[0, 1], y ∈[0, 1]
Grilla: 501 x 501 puntos
Tolerancia: 0.0001

Resolviendo la ecuación...

== RESULTADOS ACTIVIDAD 1: #pragma omp parallel for ==
Tiempo de ejecución: 263.337 segundos
Número de iteraciones: 514193
Número de threads: 32
Resultados exportados a actividad1/ejemplo_0_parallel_for_numerical.dat

Simulación completada.
```

Resultados: Actividad 1

```
==== RESULTADOS ACTIVIDAD 1: #pragma omp parallel for ===
Tiempo de ejecución: 8.383 segundos
Número de iteraciones: 16609
Número de threads: 32

==== ANÁLISIS DE ERROR ===
Error máximo: 1.07184
Error RMS: 0.545961
Resultados exportados a actividad1/ejemplo_1_parallel_for_analytical.dat
Resultados exportados a actividad1/ejemplo_1_parallel_for_numerical.dat
```

Ejemplo 1

```
==== RESULTADOS ACTIVIDAD 1: #pragma omp parallel for ===
Tiempo de ejecución: 2.674 segundos
Número de iteraciones: 5351
Número de threads: 32

==== ANÁLISIS DE ERROR ===
Error máximo: 1.11902
Error RMS: 0.632532
Resultados exportados a actividad1/ejemplo_2_parallel_for_analytical.dat
Resultados exportados a actividad1/ejemplo_2_parallel_for_numerical.dat
```

Ejemplo 2

Resultados: Actividad 1

```
==== RESULTADOS ACTIVIDAD 1: #pragma omp parallel for ===
Tiempo de ejecución: 4.535 segundos
Número de iteraciones: 9001
Número de threads: 32

==== ANÁLISIS DE ERROR ===
Error máximo: 1.09056
Error RMS: 0.394199
Resultados exportados a actividad1/ejemplo_3_parallel_for_analytical.dat
Resultados exportados a actividad1/ejemplo_3_parallel_for_numerical.dat
```

Ejemplo 3

```
==== RESULTADOS ACTIVIDAD 1: #pragma omp parallel for ===
Tiempo de ejecución: 7.267 segundos
Número de iteraciones: 14437
Número de threads: 32

==== ANÁLISIS DE ERROR ===
Error máximo: 1.67491
Error RMS: 0.808973
Resultados exportados a actividad1/ejemplo_4_parallel_for_analytical.dat
Resultados exportados a actividad1/ejemplo_4_parallel_for_numerical.dat
```

Ejemplo 4

Resultados: Actividad 1

Ejemplo	Tiempo de ejecución (s)	Número de iteraciones	Número de threads
0	263,337	514193	32
1	8,383	16609	32
2	2,674	5351	32
3	4,535	9001	32
4	7,267	14437	32

Resultados: Actividad 1

¿Hubo diferencia en el número de iteraciones?

Si, como se puede observar en los resultados obtenidos el número de iteraciones aumentaba conforme el tiempo de ejecución crecía. El caso más evidente corresponde a la ejecución del ejemplo original con fuente de tipo gaussiana.

¿Por qué es necesaria la cláusula reduction?

La cláusula reduction(max:delta) es necesaria para evitar condiciones de carrera cuando varios hilos actualizan una misma variable compartida, en este caso delta, que guarda el máximo cambio entre una iteración y la siguiente.

Actividad 2: collapse(2)

- Sustituye el bucle paralelo anterior por: `#pragma omp parallel for collapse(2) reduction(max:delta)`

```
admin@ip-172-31-21-223:~/Taller_OpenMP_Poisson/src$ ./actividad_2
== ACTIVIDAD 2: SIMULADOR DE ECUACIÓN DE POISSON 2D (collapse(2)) ==
Esta implementación utiliza #pragma omp parallel for collapse(2) reduction(max:delta)
para acelerar el anidamiento de bucles.

Directorio 'actividad2' creado exitosamente.
Ejemplos disponibles:
0 - Ejemplo Original: Término fuente gaussiano
1 - Ejemplo 1:  $\nabla^2 V = (x^2 + y^2)e^{xy}$ 
2 - Ejemplo 2:  $\nabla^2 V = 0$  (Ecuación de Laplace)
3 - Ejemplo 3:  $\nabla^2 V = 4$ 
4 - Ejemplo 4:  $\nabla^2 V = x/y + y/x$ 
5 - Ejecutar todos los ejemplos
```

Resultados: Actividad 2

Ejemplo 0

```
==== EJEMPLO 0: Término fuente gaussiano ====
Dominio: x ∈[0, 1], y ∈[0, 1]
Grilla: 501 x 501 puntos
Tolerancia: 0.0001

Resolviendo la ecuación...

==== RESULTADOS ACTIVIDAD 2: collapse(2) ====
Tiempo de ejecución: 304.371 segundos
Número de iteraciones: 514189
Número de threads: 32
Resultados exportados a actividad2/ejemplo_0_collapse2_numerical.dat

Simulación completada.
```

Resultados: Actividad 2

```
==> RESULTADOS ACTIVIDAD 2: collapse(2) ==>
Tiempo de ejecución: 8.812 segundos
Número de iteraciones: 16610
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.07175
Error RMS: 0.545927
Resultados exportados a actividad2/ejemplo_1_collapse2_analytical.dat
Resultados exportados a actividad2/ejemplo_1_collapse2_numerical.dat
```

Ejemplo 1

```
==> RESULTADOS ACTIVIDAD 2: collapse(2) ==>
Tiempo de ejecución: 3.148 segundos
Número de iteraciones: 5351
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.11902
Error RMS: 0.632537
Resultados exportados a actividad2/ejemplo_2_collapse2_analytical.dat
Resultados exportados a actividad2/ejemplo_2_collapse2_numerical.dat
```

Ejemplo 2

Resultados: Actividad 2

```
==> RESULTADOS ACTIVIDAD 2: collapse(2) ==>
Tiempo de ejecución: 4.531 segundos
Número de iteraciones: 9001
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.09055
Error RMS: 0.394205
Resultados exportados a actividad2/ejemplo_3_collapse2_analytical.dat
Resultados exportados a actividad2/ejemplo_3_collapse2_numerical.dat
```

Ejemplo 3

```
==> RESULTADOS ACTIVIDAD 2: collapse(2) ==>
Tiempo de ejecución: 7.326 segundos
Número de iteraciones: 14437
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.67498
Error RMS: 0.80898
Resultados exportados a actividad2/ejemplo_4_collapse2_analytical.dat
Resultados exportados a actividad2/ejemplo_4_collapse2_numerical.dat
```

Ejemplo 4

Resultados: Actividad 2

¿Mejoró respecto a la versión anterior?

Ejemplo	Tiempo Actividad 1	Tiempo Actividad 2	Diferencia
0	263,337 s	304,371 s	+41 s (más lento)
1	8,383 s	8,812 s	+0,43 s
2	2,674 s	3,148 s	+0,47 s
3	4,535 s	4,531 s	= igual
4	7,267 s	7,326 s	+0,06 s

Cuadro 1: Comparación de tiempos entre Actividad 1 y Actividad 2

Resultados: Actividad 2

¿Qué hace collapse(2)?

La cláusula collapse(n) en OpenMP combinan bucles anidados en uno solo para que la iteración total pueda distribuirse entre los hilos de manera más equilibrada.

¿Es siempre recomendable su uso?

No. No siempre es recomendable. Depende de varios factores:

Útil cuando los bucles anidados tienen mucho trabajo y pocos hilos lo aprovechan bien.

Actividad 3: sections

- Paraleliza `initialize_grid` y `poisson_source` con `sections`.

```
==== ACTIVIDAD 3: SIMULADOR DE ECUACIÓN DE POISSON 2D (sections) ====
Esta implementación utiliza #pragma omp parallel sections
para ejecutar funciones independientes en paralelo.

Ejemplos disponibles:
0 - Ejemplo Original: Término fuente gaussiano
1 - Ejemplo 1:  $\nabla^2 V = (x^2 + y^2)e^{xy}$ 
2 - Ejemplo 2:  $\nabla^2 V = 0$  (Ecuación de Laplace)
3 - Ejemplo 3:  $\nabla^2 V = 4$ 
4 - Ejemplo 4:  $\nabla^2 V = x/y + y/x$ 
5 - Ejecutar todos los ejemplos

Seleccione una opción (0-5): 0
```

Resultados: Actividad 3

Ejemplo 0

```
== INICIALIZACIÓN PARALELA CON SECTIONS ==
Thread Thread 420 ejecutando calculate_source_term ejecutando initialize_boundary_conditions
Tiempo de inicialización paralela (sections): 0.011 segundos
Resolviendo la ecuación...
== RESULTADOS ACTIVIDAD 3: sections ==
Tiempo de ejecución total: 266.212 segundos
Número de iteraciones: 514196
Número de threads: 32
Resultados exportados a actividad3/ejemplo_0_sections_numerical.dat
Simulación completada.
```

Resultados: Actividad 3

```
==> RESULTADOS ACTIVIDAD 3: sections ==>
Tiempo de ejecución total: 8.338 segundos
Número de iteraciones: 16609
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.07184
Error RMS: 0.545961
Resultados exportados a actividad3/ejemplo_1_sections_analytical.dat
Resultados exportados a actividad3/ejemplo_1_sections_numerical.dat
```

Ejemplo 1

```
==> RESULTADOS ACTIVIDAD 3: sections ==>
Tiempo de ejecución total: 2.724 segundos
Número de iteraciones: 5351
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.11902
Error RMS: 0.632532
Resultados exportados a actividad3/ejemplo_2_sections_analytical.dat
Resultados exportados a actividad3/ejemplo_2_sections_numerical.dat
```

Ejemplo 2

Resultados: Actividad 3

```
==> RESULTADOS ACTIVIDAD 3: sections ==>
Tiempo de ejecución total: 4.567 segundos
Número de iteraciones: 9001
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.09056
Error RMS: 0.394199
Resultados exportados a actividad3/ejemplo_3_sections_analytical.dat
Resultados exportados a actividad3/ejemplo_3_sections_numerical.dat
```

Ejemplo 3

```
==> RESULTADOS ACTIVIDAD 3: sections ==>
Tiempo de ejecución total: 7.192 segundos
Número de iteraciones: 14437
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.67491
Error RMS: 0.808973
Resultados exportados a actividad3/ejemplo_4_sections_analytical.dat
Resultados exportados a actividad3/ejemplo_4_sections_numerical.dat
```

Ejemplo 4

Resultados: Actividad 3

¿Tuviste que reordenar las funciones? ¿Por qué?

Sí. Fue necesario reorganizar el código para asegurar que las funciones fueran seguras en un entorno paralelo. Esto incluyó evitar dependencias globales y asegurar que cada hilo tuviera acceso a su propia copia de los datos necesarios.

Recomendaciones:

- Separar claramente las etapas: lectura, inicialización, cálculo, postprocesamiento, etc.
- Colocar primero las funciones que sí se pueden parallelizar y después las que no.
- Agrupar loops que se pueden parallelizar, minimizando dependencias.

¿Vale la pena parallelizar funciones tan rápidas?

No. Cuando las funciones son muy rápidas o hacen operaciones simples, la sobrecarga de gestionarlas en paralelo puede superar el beneficio. Es más eficiente mantenerlas dentro de un bucle paralelizado, en lugar de parallelizarlas individualmente.

Actividad 4: **parallel** y **for** separados

Usa **#pragma omp parallel {#pragma omp for reduction(max:delta)}**

Añade variantes con **schedule(static)** y **schedule(dynamic)**.

```
==== ACTIVIDAD 4: SIMULADOR DE ECUACIÓN DE POISSON 2D ====
Control explícito de región paralela con diferentes estrategias de scheduling
```

Esta implementación utiliza:

- **#pragma omp parallel**
- **#pragma omp for reduction(max:delta)**
- **schedule(static)** y **schedule(dynamic)**

Ejemplos disponibles:

- 0 - Ejemplo Original: Término fuente gaussiano
- 1 - Ejemplo 1: $\nabla^2 V = (x^2 + y^2)e^{xy}$
- 2 - Ejemplo 2: $\nabla^2 V = 0$ (Ecuación de Laplace)
- 3 - Ejemplo 3: $\nabla^2 V = 4$
- 4 - Ejemplo 4: $\nabla^2 V = x/y + y/x$
- 5 - Ejecutar todos los ejemplos con todas las estrategias

Seleccione una opción (0-5): □

Resultados: Actividad 4

Ejemplo 0

static

```
Resolviendo la ecuación con control explícito...
Ejecutando con estrategia: STATIC
```

```
==> RESULTADOS ACTIVIDAD 4: STATIC ===
Tiempo de ejecución: 269.689 segundos
Número de iteraciones: 514193
Número de threads: 32
Resultados exportados a actividad4/ejemplo_0_static
Simulación completada.
```

dynamic

```
Resolviendo la ecuación con control explícito...
Ejecutando con estrategia: DYNAMIC
```

```
==> RESULTADOS ACTIVIDAD 4: DYNAMIC ===
Tiempo de ejecución: 474.662 segundos
Número de iteraciones: 548156
Número de threads: 32
Resultados exportados a actividad4/ejemplo_0_dynamic
Simulación completada.
```

Resultados: Actividad 4

Ejemplo 1

```
==> RESULTADOS ACTIVIDAD 4: STATIC ==>
Tiempo de ejecución: 10.039 segundos
Número de iteraciones: 16609
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.07184
Error RMS: 0.545961
```

static

```
==> RESULTADOS ACTIVIDAD 4: DYNAMIC ==>
Tiempo de ejecución: 15.991 segundos
Número de iteraciones: 18719
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 0.964556
Error RMS: 0.484009
```

dynamic

Ejemplo 2

```
==> RESULTADOS ACTIVIDAD 4: STATIC ==>
Tiempo de ejecución: 2.783 segundos
Número de iteraciones: 5351
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.11902
Error RMS: 0.632532
```

static

```
==> RESULTADOS ACTIVIDAD 4: DYNAMIC ==>
Tiempo de ejecución: 5.792 segundos
Número de iteraciones: 6846
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.07331
Error RMS: 0.597428
```

dynamic

Resultados: Actividad 4

Ejemplo 3

```
==> RESULTADOS ACTIVIDAD 4: STATIC ==>
Tiempo de ejecución: 4.669 segundos
Número de iteraciones: 9001
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.09056
Error RMS: 0.394199
```

static

```
==> RESULTADOS ACTIVIDAD 4: DYNAMIC ==>
Tiempo de ejecución: 10.25 segundos
Número de iteraciones: 12121
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 0.932891
Error RMS: 0.345932
```

dynamic

Ejemplo 4

```
==> RESULTADOS ACTIVIDAD 4: STATIC ==>
Tiempo de ejecución: 7.509 segundos
Número de iteraciones: 14437
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.67491
Error RMS: 0.808973
```

static

```
==> RESULTADOS ACTIVIDAD 4: DYNAMIC ==>
Tiempo de ejecución: 15.944 segundos
Número de iteraciones: 18811
Número de threads: 32

==> ANÁLISIS DE ERROR ==
Error máximo: 1.40563
Error RMS: 0.675962
```

dynamic

Resultados: Actividad 4

Ejemplo	Iteraciones	static (s)	dynamic (s)	Comentario
0	514193 vs 548156	269.689	474.662	Muchísimas iteraciones, dynamic sufre por overhead.
1	16609 vs 18719	10.339	15.991	Similar efecto.
2	5351 vs 6846	2.783	5.792	Pocas iteraciones, static gana.
3	9001 vs 12121	4.669	10.25	Carga relativamente balanceada.
4	14437 vs 18811	7.509	15.944	Mismo patrón.

Cuadro 2: Comparación de rendimiento entre planificación static y dynamic

Actividad 5: **nowait, barrier, single, critical**

Usa nowait en bucles independientes.

Imprime inicio de cada iteración con **#pragma omp single**.

Agrega sección artificial crítica con **#pragma omp critical**.

```
==== ACTIVIDAD 5: CONTROL DE SINCRONIZACIÓN OPENMP ====
Esta implementación utiliza las directivas:
• #pragma omp nowait - Evita sincronización innecesaria
• #pragma omp barrier - Sincronización explícita
• #pragma omp single - Ejecución por un solo thread
• #pragma omp critical - Sección crítica para acceso exclusivo

Directorio 'actividad5' creado exitosamente.
Ejemplos disponibles:
0 - Ejemplo Original: Término fuente gaussiano
1 - Ejemplo 1:  $\nabla^2 V = (x^2 + y^2)e^{xy}$ 
2 - Ejemplo 2:  $\nabla^2 V = 0$  (Ecuación de Laplace)
3 - Ejemplo 3:  $\nabla^2 V = 4$ 
4 - Ejemplo 4:  $\nabla^2 V = x/y + y/x$ 
5 - Ejecutar todos los ejemplos
```

Resultados: Actividad 5

Ejemplo 0

```
Seleccione una opción (0-5): 0
```

```
== EJEMPLO 0: Término fuente gaussiano ==
```

```
Dominio:  $x \in [0, 1]$ ,  $y \in [0, 1]$ 
```

```
Grilla: 501 x 501 puntos
```

```
Tolerancia: 0.0001
```

```
== RESULTADOS ACTIVIDAD 5: Control de Sincronización ==
```

```
Tiempo de ejecución: 263.83 segundos
```

```
Número de iteraciones: 514162
```

```
Resultados exportados a actividad5/ejemplo_0_sync_numerical.dat
```

```
Simulación completada.
```

Resultados: Actividad 5

```
==> RESULTADOS ACTIVIDAD 5: Control de Sincronización == Ejemplo 1
Tiempo de ejecución: 9.752 segundos
Número de iteraciones: 16610

==> ANÁLISIS DE ERROR ==
Error máximo: 1.07175
Error RMS: 0.545927
Resultados exportados a actividad5/ejemplo_1_sync_analytical.dat
Resultados exportados a actividad5/ejemplo_1_sync_numerical.dat
```

```
==> RESULTADOS ACTIVIDAD 5: Control de Sincronización == Ejemplo 2
Tiempo de ejecución: 2.682 segundos
Número de iteraciones: 5351

==> ANÁLISIS DE ERROR ==
Error máximo: 1.11902
Error RMS: 0.632537
Resultados exportados a actividad5/ejemplo_2_sync_analytical.dat
Resultados exportados a actividad5/ejemplo_2_sync_numerical.dat
```

Resultados: Actividad 5

```
==== RESULTADOS ACTIVIDAD 5: Control de Sincronización === Ejemplo 3
Tiempo de ejecución: 4.569 segundos
Número de iteraciones: 9001

==== ANÁLISIS DE ERROR ===
Error máximo: 1.09055
Error RMS: 0.394205
Resultados exportados a actividad5/ejemplo_3_sync_analytical.dat
Resultados exportados a actividad5/ejemplo_3_sync_numerical.dat
```

```
==== RESULTADOS ACTIVIDAD 5: Control de Sincronización === Ejemplo 4
Tiempo de ejecución: 7.294 segundos
Número de iteraciones: 14437

==== ANÁLISIS DE ERROR ===
Error máximo: 1.67498
Error RMS: 0.80898
Resultados exportados a actividad5/ejemplo_4_sync_analytical.dat
Resultados exportados a actividad5/ejemplo_4_sync_numerical.dat
```

Resultados: Actividad 5

Ejemplo	Tiempo de ejecución (s)	Número de iteraciones
0	263,83	514162
1	9,752	16610
2	2,682	5351
3	4,569	9001
4	7,294	14437

Resultados: Actividad 5

¿Hubo cambio con respecto al uso por defecto?

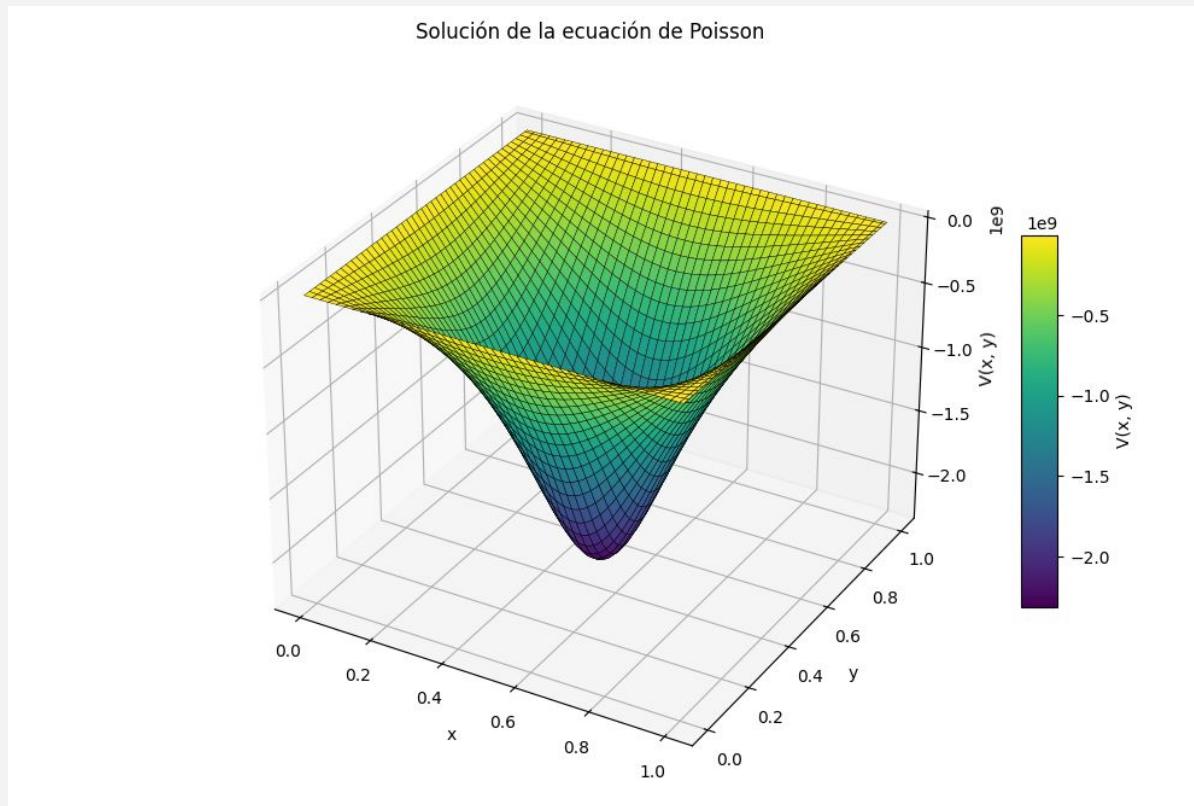
Sí, pero leve. El uso por defecto de OpenMP introduce barreras implícitas al final de cada bucle for y cada región paralela, lo cual sincroniza todos los hilos antes de continuar. Al usar nowait, se elimina la sincronización automática, permitiendo que los hilos que terminan un bucle puedan continuar con otro trabajo sin esperar.

¿critical ralentizó la ejecución? ¿En qué contexto sería útil?

Sí, ralentizó ligeramente. Es útil cuando varios hilos acceden a una sección que debe ejecutarse de forma exclusiva. Aquí se usa para incrementar el contador convergence_checks de forma segura cada 50 iteraciones, sin riesgo de condición de carrera. Ahora bien, si múltiples hilos ejecutarán esta línea simultáneamente sin critical, podría perderse la actualización de algunos hilos.

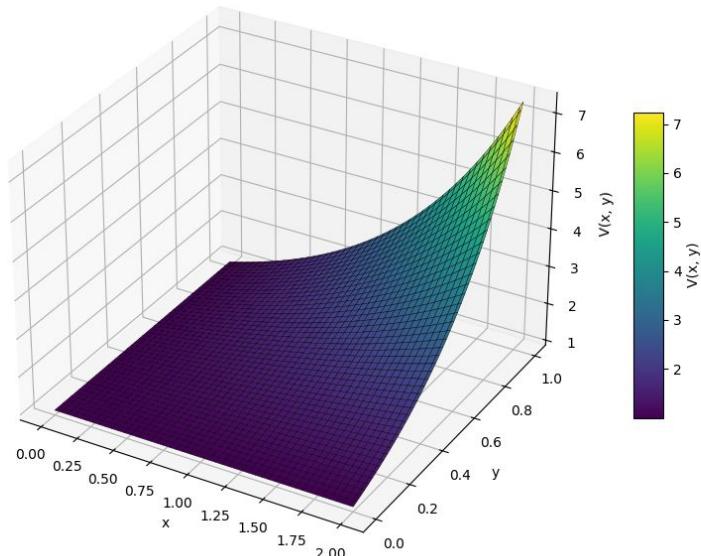
Gráficas Obtenidas con paralelización

Ejemplo 0: Término Fuente Gaussiano 300x300



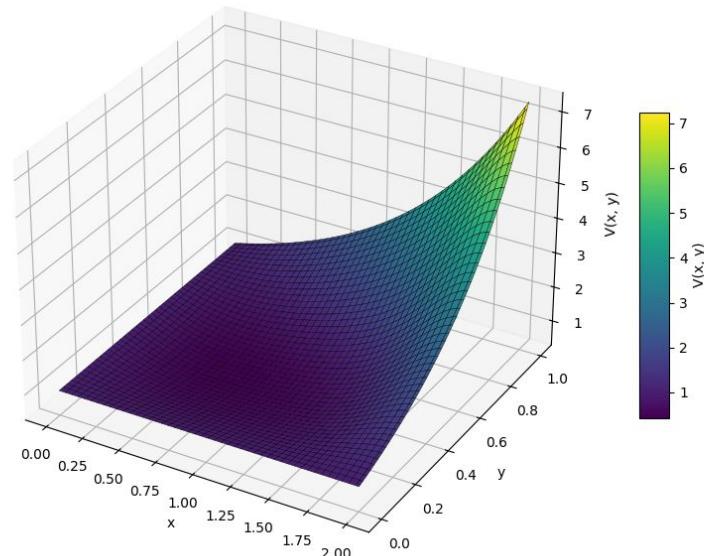
Ejemplo 1: Término Fuente Exponencial 300x300

Solución de la ecuación de Poisson



Analítico

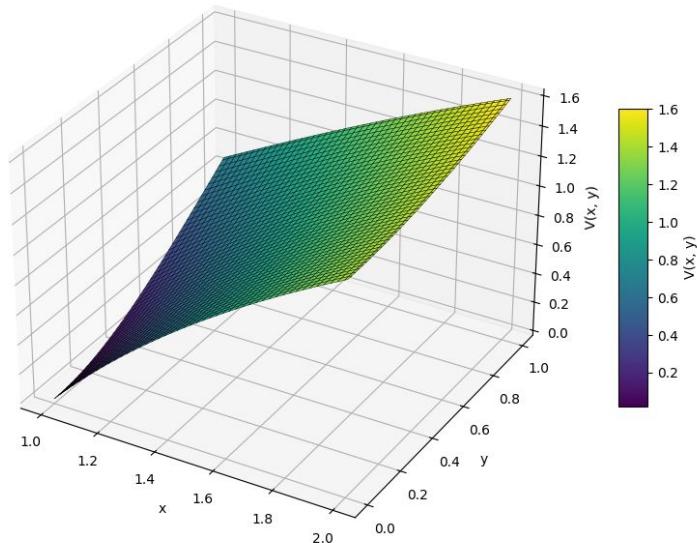
Solución de la ecuación de Poisson



Numérico

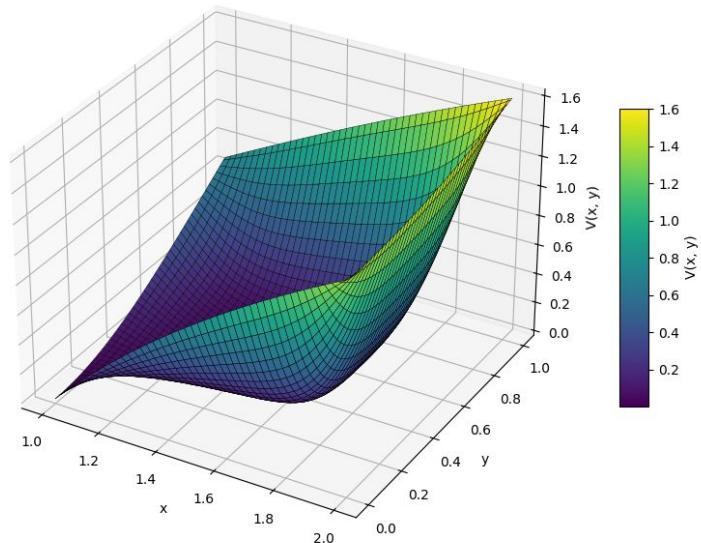
Ejemplo 2: Ecuación de Laplace 300x300

Solución de la ecuación de Poisson



Analítico

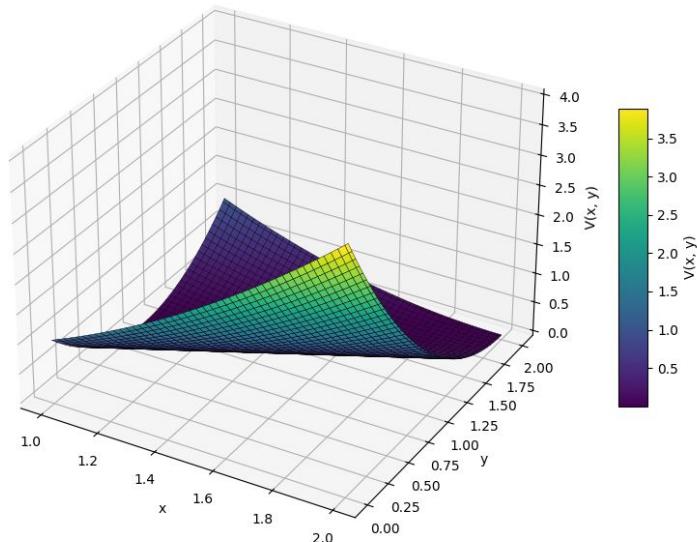
Solución de la ecuación de Poisson



Numérico

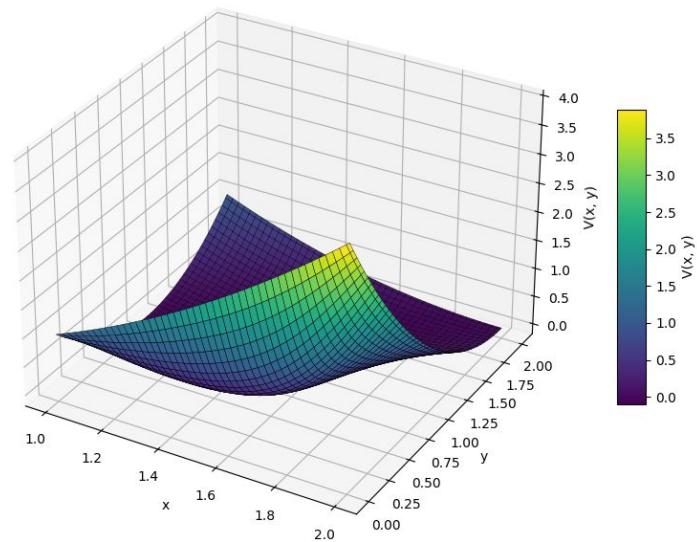
Ejemplo 3: Término Fuente Constante 300x300

Solución de la ecuación de Poisson



Analítico

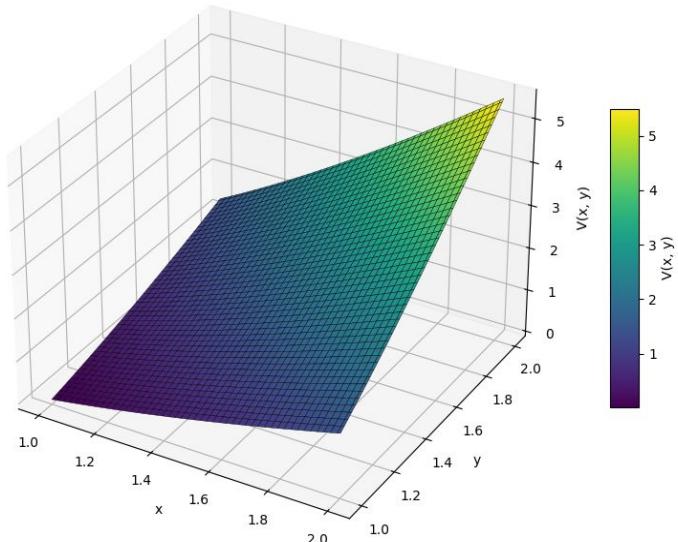
Solución de la ecuación de Poisson



Numérico

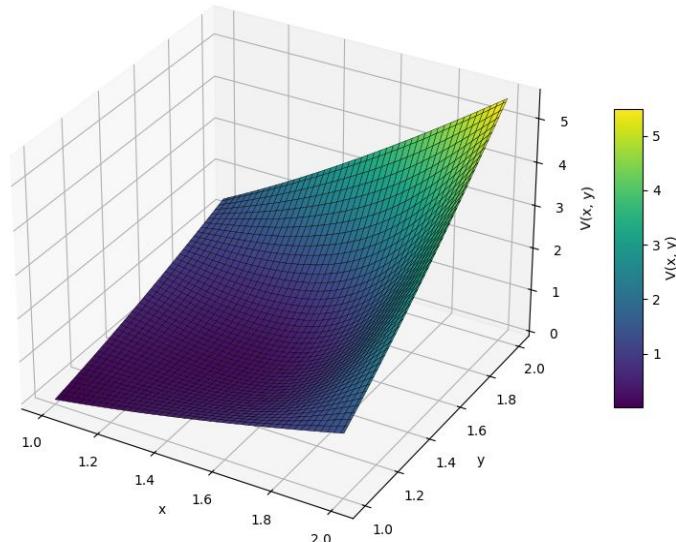
Ejemplo 4: Término Fuente Racional 300x300

Solución de la ecuación de Poisson



Analítico

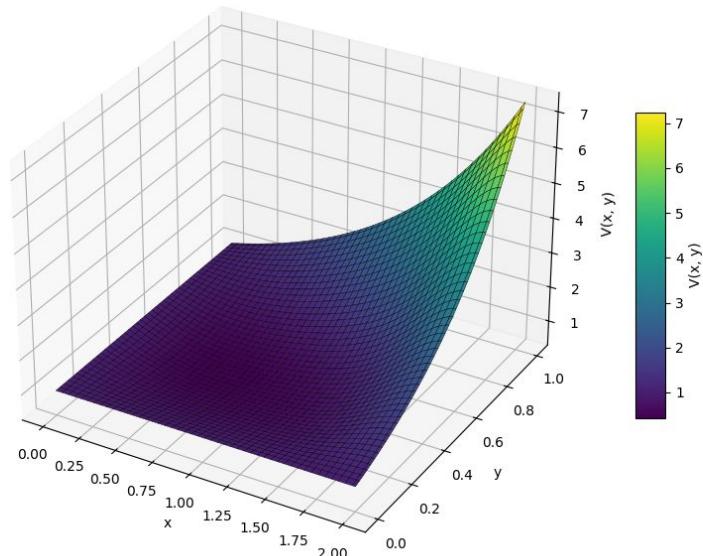
Solución de la ecuación de Poisson



Numérico

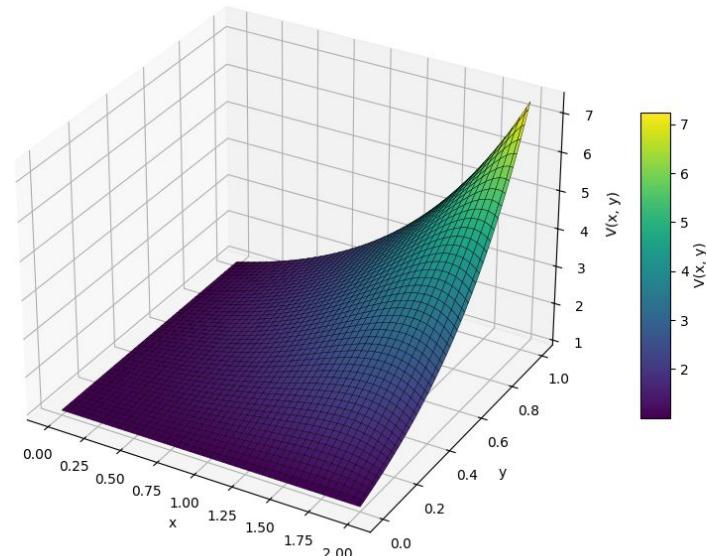
Ejemplo 1: Término Fuente Exponencial 500x500

Solución de la ecuación de Poisson



Numérico

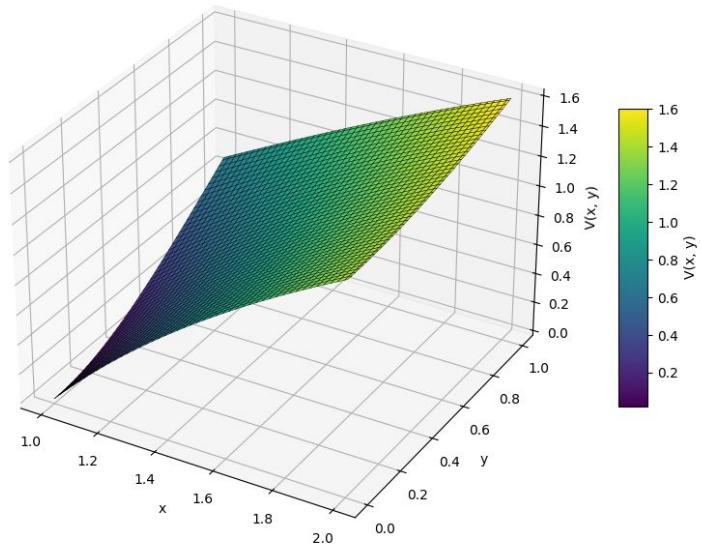
Solución de la ecuación de Poisson



Numérico

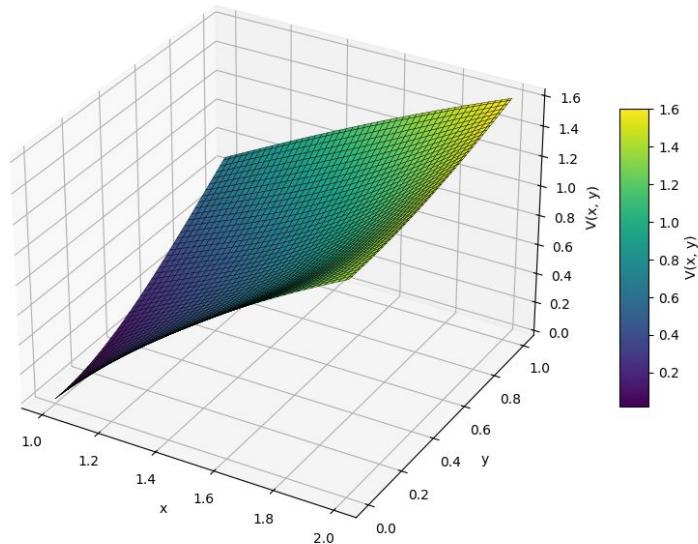
Ejemplo 2: Ecuación de Laplace 500x500

Solución de la ecuación de Poisson



Numérico

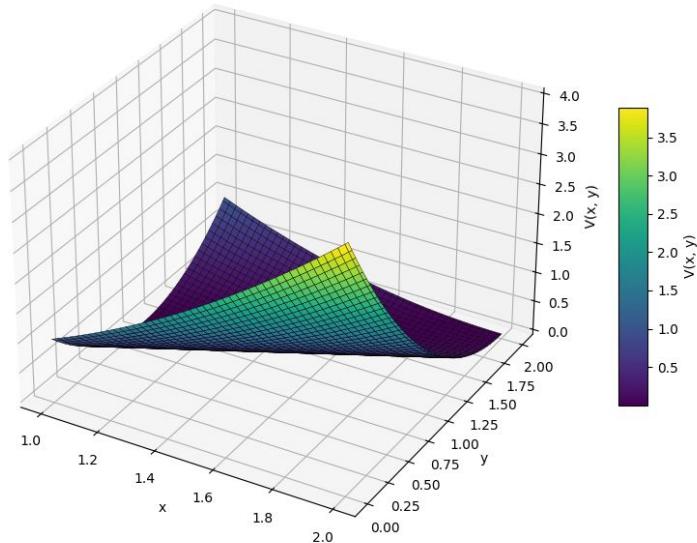
Solución de la ecuación de Poisson



Numérico

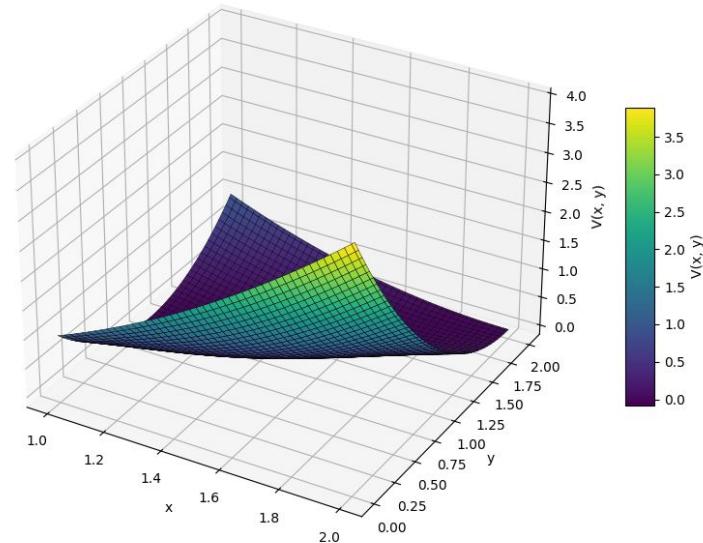
Ejemplo 3: Término Fuente Constante 500x500

Solución de la ecuación de Poisson



Numérico

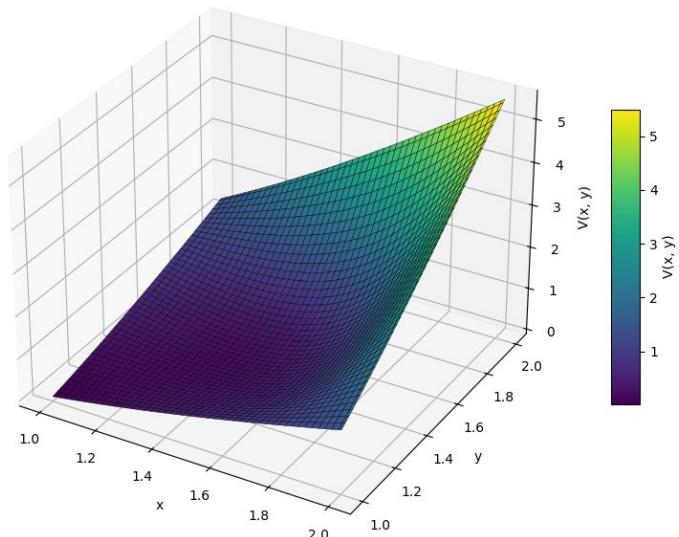
Solución de la ecuación de Poisson



Numérico

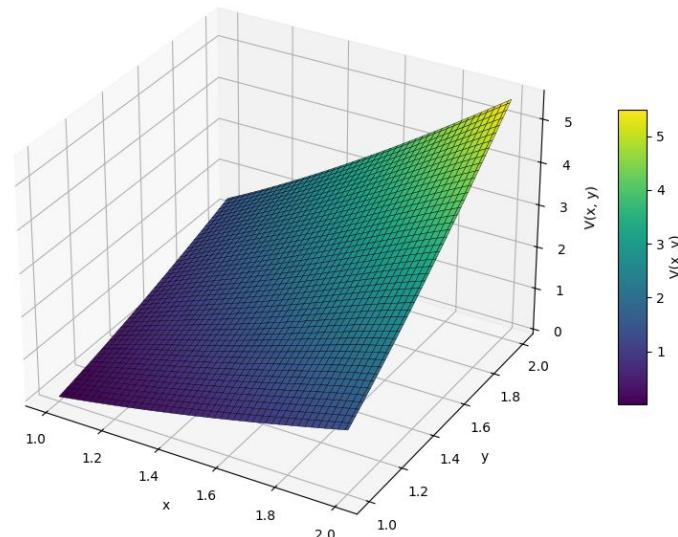
Ejemplo 4: Término Fuente Racional 500x500

Solución de la ecuación de Poisson



Numérico

Solución de la ecuación de Poisson



Numérico

Comparación de estrategias

Ejemplo 0

Versión	Directiva usada	Tiempo (s)	Iteraciones	Observaciones
Secuencial		693,216	246519	Mayor tiempo de ejecución
Paralelo básico	#pragma omp parallel reduction(max:delta)	263,337	514193	Versión más rápida y simple
Colapsado de bucles	#pragma omp parallel for collapse(2) reduction(max:delta)	304,371	514189	Tercera versión más lenta, poco beneficio
Inicialización y fuente en paralelo	#pragma omp parallel sections, #pragma omp parallel section	266,212	514196	Tercera versión más eficiente, eficiencia de sections
Control explícito + schedule	#pragma omp parallel #pragma omp for schedule(static/dynamic) reduction(max:delta)	static: 269,689 dynamic: 474,662	static: 514193 dynamic: 548156	Mayor tiempo de ejecución e iteraciones con dynamic, más eficiente en static
Control de sincronización	#pragma omp parallel #pragma omp for nowait #pragma omp for collapse(2) nowait #pragma omp for collapse(2) reduction(max:delta) nowait #pragma omp critical(convergence_counter), #pragma omp barrier	263,83	514162	Segunda versión más eficiente y con menor número de iteraciones

Comparación de estrategias

Ejemplo 1

Versión	Directiva usada	Tiempo (s)	Iteraciones	Observaciones
Secuencial		152,419	54646	Mayor tiempo de ejecución y de iteraciones
Paralelo básico	#pragma omp parallel reduction(max:delta)	8,383	16609	Versión más simple y la segunda más eficiente
Colapsado de bucles	#pragma omp parallel for collapse(2) reduction(max:delta)	8,812	16610	Versión con buena eficiencia
Inicialización y fuente en paralelo	#pragma omp parallel sections, #pragma omp parallel section	8,338	16609	Versión más eficiente con el uso de sections
Control explícito + schedule	#pragma omp parallel #pragma omp for schedule(static/dynamic) reduction(max:delta)	static: 10,039 dynamic: 15,991	static: 16609 dynamic: 18719	Mayor tiempo de ejecución e iteraciones con dynamic, más eficiente en static
Control de sincronización	#pragma omp parallel #pragma omp for nowait #pragma omp for collapse(2) nowait #pragma omp for collapse(2) reduction(max:delta) nowait #pragma omp critical(convergence_counter), #pragma omp barrier	9,752	16610	Segunda versión más lenta pero bastante eficiente en comparación al código secuencial

Comparación de estrategias

Ejemplo 2

Versión	Directiva usada	Tiempo (s)	Iteraciones	Observaciones
Secuencial		129,502	46484	Menos eficiente y mayor número de iteraciones
Paralelo básico	#pragma omp parallel reduction(max:delta)	2,674	5351	Versión más eficiente y simple
Colapsado de bucles	#pragma omp parallel for collapse(2) reduction(max:delta)	3,148	5351	Eficiente pero no tanto en comparación a las otras versiones paralelizadas
Inicialización y fuente en paralelo	#pragma omp parallel sections, #pragma omp parallel section	2,724	5351	Eficiencia en el uso de sections
Control explícito + schedule	#pragma omp parallel #pragma omp for schedule(static/dynamic) reduction(max:delta)	static: 2,783 dynamic: 6846	static: 5351 dynamic: 18719	Segunda versión menos eficiente con dynamic, mejor opción static
Control de sincronización	#pragma omp parallel #pragma omp for nowait #pragma omp for collapse(2) nowait #pragma omp for collapse(2) reduction(max:delta) nowait #pragma omp critical(convergence_counter), #pragma omp barrier	2,682	5351	Segunda versión más eficiente pero una de las más complejas

Comparación de estrategias

Ejemplo 3

Versión	Directiva usada	Tiempo (s)	Iteraciones	Observaciones
Secuencial		113,963	40895	Menos eficiente y mayor número de iteraciones
Paralelo básico	#pragma omp parallel reduction(max:delta)	4,535	9001	Segunda versión más eficiente, muy cercano al collapse
Colapsado de bucles	#pragma omp parallel for collapse(2) reduction(max:delta)	4,531	9001	Versión más eficiente
Inicialización y fuente en paralelo	#pragma omp parallel sections, #pragma omp parallel section	4,567	9001	Eficiencia en el uso de sections
Control explícito + schedule	#pragma omp parallel #pragma omp for schedule(static/dynamic) reduction(max:delta)	static: 4,669 dynamic: 10,25	static: 9001 dynamic: 12121	Segunda versión menos eficiente con dynamic, mejor opción static
Control de sincronización	#pragma omp parallel #pragma omp for nowait #pragma omp for collapse(2) nowait #pragma omp for collapse(2) reduction(max:delta) nowait #pragma omp critical(convergence_counter), #pragma omp barrier	4,569	9001	Eficiencia similar al caso de sections pero una versión más compleja

Comparación de estrategias

Ejemplo 4

Versión	Directiva usada	Tiempo (s)	Iteraciones	Observaciones
Secuencial		150,338	53975	Menos eficiente y mayor número de iteraciones
Paralelo básico	#pragma omp parallel reduction(max:delta)	7,267	14437	Segunda versión más eficiente, muy cercano al sections
Colapsado de bucles	#pragma omp parallel for collapse(2) reduction(max:delta)	7,326	14437	Versión eficiente
Inicialización y fuente en paralelo	#pragma omp parallel sections, #pragma omp parallel section	7,192	14437	Versión más eficiente en el uso de sections
Control explícito + schedule	#pragma omp parallel #pragma omp for schedule(static/dynamic) reduction(max:delta)	static: 7,509 dynamic: 15,949	static: 14437 dynamic: 12121	Según versión menos eficiente con dynamic, mejor opción static
Control de sincronización	#pragma omp parallel #pragma omp for nowait #pragma omp for collapse(2) nowait #pragma omp for collapse(2) reduction(max:delta) nowait #pragma omp critical(convergence_counter), #pragma omp barrier	7,292	14437	Eficiencia similar al caso de parallel simple pero una versión más compleja

Conclusiones

1. ¿Cuál fue la versión más rápida?

Ejemplo	Versión	Tiempo (s)	Iteraciones
Ejemplo 0	Paralelo básico	263,337	514193
Ejemplo 1	Inicialización y fuente en paralelo	8,338	16609
Ejemplo 2	Paralelo básico	2,674	5351
Ejemplo 3	Colapso de bucles	4,531	9001
Ejemplo 4	Inicialización y fuente en paralelo	7,192	14437

Conclusiones

2. ¿Qué directiva es más fácil de usar?

Respuesta: **#pragma omp parallel reduction(max:delta)** (Paralelo básico) es la más práctica y eficiente.

Ventajas técnicas:

- **Simplicidad:** Una sola directiva
- **Gestión automática:** Manejo automático de variables compartidas y privadas
- **Reducción integrada:** Operación de reducción optimizada por el compilador
- **Portabilidad:** Compatible con todas las implementaciones OpenMP
- **Debugging:** Menor complejidad para depuración
- **Mantenimiento:** Código más legible y mantenible

Conclusiones

3. ¿Hubo directivas que empeoraron el rendimiento? ¿Por qué?

Respuesta: Sí, ciertas directivas, especialmente schedule(dynamic), empeoraron el rendimiento comparado con otras versiones paralelas.

Causa del bajo rendimiento con schedule(dynamic):

- La directiva dynamic divide las iteraciones de forma más flexible pero introduce sobrecarga adicional de planificación, ya que las tareas se asignan dinámicamente en tiempo de ejecución.
- Este enfoque es útil cuando las iteraciones son desbalanceadas, pero si todas tienen tiempos similares, esta flexibilidad se convierte en un gasto innecesario.

Otras directivas con menor impacto negativo:

Las combinaciones con #pragma omp critical y #pragma omp barrier también pueden degradar el rendimiento si no se usan con cuidado, ya que sincronizan o serializan partes del código que podrían ejecutarse en paralelo.

Actividad 6: Número de iteraciones con núcleo compartido

En esta actividad, vas a modificar la función solve_poisson para contar el número total de iteraciones del bucle externo (las actualizaciones completas de la grilla) utilizando una variable global o compartida.

```
==> ACTIVIDAD 6: CONTADOR DE ITERACIONES CON CRITICAL Y ATOMIC ==>
Directorio 'actividad6' creado exitosamente.

Ejemplos disponibles:
0 - Ejemplo Original: Término fuente gaussiano
1 - Ejemplo 1:  $\nabla^2 V = (x^2 + y^2)e^{xy}$ 
2 - Ejemplo 2:  $\nabla^2 V = 0$  (Ecuación de Laplace)
3 - Ejemplo 3:  $\nabla^2 V = 4$ 
4 - Ejemplo 4:  $\nabla^2 V = x/y + y/x$ 
5 - Ejecutar todos los ejemplos
```

Resultados: Actividad 6

Ejemplo 0

```
== ACTIVIDAD 6: EJEMPLO 0 ==
Descripción: Término fuente gaussiano
Número de threads disponibles: 32

== PRUEBA CON #pragma omp critical ==
Tiempo de ejecución: 267.146 segundos
Número de iteraciones: 514162

== PRUEBA CON #pragma omp atomic ==
Tiempo de ejecución: 267.45 segundos
Número de iteraciones: 514162

== COMPARACIÓN DE RENDIMIENTO ==
Diferencia de tiempo: 0.304 segundos
CRITICAL es 0.113666% más rápido que ATOMIC
Verificación: Ambos métodos convergen en 514162 iteraciones: SÍ
```

Resultados: Actividad 6

Ejemplo 1

```
== ACTIVIDAD 6: EJEMPLO 1 ==
Descripción:  $\nabla^2 V = (x^2 + y^2)e^{xy}$ 
Número de threads disponibles: 32

== PRUEBA CON #pragma omp critical ==
Tiempo de ejecución: 8.404 segundos
Número de iteraciones: 16610

== PRUEBA CON #pragma omp atomic ==
Tiempo de ejecución: 8.468 segundos
Número de iteraciones: 16610

== COMPARACIÓN DE RENDIMIENTO ==
Diferencia de tiempo: 0.064 segundos
CRITICAL es 0.755786% más rápido que ATOMIC
```

Ejemplo 2

```
== ACTIVIDAD 6: EJEMPLO 2 ==
Descripción:  $\nabla^2 V = 0$  (Laplace)
Número de threads disponibles: 32

== PRUEBA CON #pragma omp critical ==
Tiempo de ejecución: 2.7 segundos
Número de iteraciones: 5351

== PRUEBA CON #pragma omp atomic ==
Tiempo de ejecución: 2.709 segundos
Número de iteraciones: 5351

== COMPARACIÓN DE RENDIMIENTO ==
Diferencia de tiempo: 0.009 segundos
CRITICAL es 0.332226% más rápido que ATOMIC
```

Resultados: Actividad 6

Ejemplo 3

```
== ACTIVIDAD 6: EJEMPLO 3 ==
Descripción:  $\nabla^2 V = 4$ 
Número de threads disponibles: 32

== PRUEBA CON #pragma omp critical ==
Tiempo de ejecución: 4.519 segundos
Número de iteraciones: 9001

== PRUEBA CON #pragma omp atomic ==
Tiempo de ejecución: 4.525 segundos
Número de iteraciones: 9001

== COMPARACIÓN DE RENDIMIENTO ==
Diferencia de tiempo: 0.006 segundos
CRITICAL es 0.132597% más rápido que ATOMIC
```

Ejemplo 4

```
== ACTIVIDAD 6: EJEMPLO 4 ==
Descripción:  $\nabla^2 V = x/y + y/x$ 
Número de threads disponibles: 32

== PRUEBA CON #pragma omp critical ==
Tiempo de ejecución: 7.462 segundos
Número de iteraciones: 14437

== PRUEBA CON #pragma omp atomic ==
Tiempo de ejecución: 7.353 segundos
Número de iteraciones: 14437

== COMPARACIÓN DE RENDIMIENTO ==
Diferencia de tiempo: 0.109 segundos
ATOMIC es 1.46073% más rápido que CRITICAL
```

Conclusiones

¿Qué diferencias observas en el rendimiento entre usar critical y atomic?

¿En qué casos sería preferible una sobre la otra?

Ejemplo	Descripción	Tiempo (segundos)		Diferencia (s)	Iteraciones	Ganado
		Critical	Atomic			
0	Término fuente gaussiano	267.146	267.450	+0.304	514162	Critical
1	$\nabla^2 V = (x^2 + y^2)e^{xy}$	8.404	8.468	+0.064	16610	Critical
2	$\nabla^2 V = 0$ (Laplace)	2.700	2.709	+0.009	5351	Critical
3	$\nabla^2 V = 4$	4.519	4.525	+0.006	9001	Critical
4	$\nabla^2 V = x/y + y/x$	7.462	7.353	-0.109	14437	Atomic
Promedio		58.046	58.101	+0.055	111912	Critical

Conclusiones

¿Qué diferencias observas en el rendimiento entre usar critical y atomic?

¿En qué casos sería preferible una sobre la otra?

Análisis de las Diferencias

1. Magnitud de las Diferencias

Las diferencias de rendimiento son muy pequeñas en todos los casos (menos del 2%), lo que indica que:

- Ambas directivas tienen overhead similar para esta aplicación específica
- El cuello de botella principal no está en la sincronización sino en el cálculo numérico
- La granularidad del trabajo protegido es adecuada para ambos enfoques

2. Patrón General Observado

En 4 de 5 ejemplos, critical fue ligeramente más rápido

Solo en el Ejemplo 4, atomic mostró una ventaja notable (1.46%)

Las diferencias son tan pequeñas que podrían estar dentro del margen de error experimental

¿En qué casos sería preferible una sobre la otra?

Cuándo usar #pragma omp critical:

- **Operaciones Complejas:** Cuando necesitas proteger múltiples líneas de código o operaciones complejas
- **Flexibilidad:** Permite proteger cualquier tipo de operación, no solo operaciones atómicas simples
- **Múltiples Variables:** Cuando necesitas actualizar varias variables de forma coherente
- **Debugging:** Más fácil de debuggear y entender el flujo de ejecución

8 hilos

Actividad 7: Exploración con tareas (task)

En esta actividad te proponemos experimentar con la directiva task de OpenMP para dividir el dominio en bloques y asignar tareas a distintos hilos.

```
==> ACTIVIDAD 7: ECUACIÓN DE POISSON 2D CON TASKS ==>

Directorio 'actividad7' creado exitosamente.
Ejemplos disponibles:
0 - Ejemplo Original: Término fuente gaussiano
1 - Ejemplo 1:  $\nabla^2 V = (x^2 + y^2)e^{xy}$ 
2 - Ejemplo 2:  $\nabla^2 V = 0$  (Ecuación de Laplace)
3 - Ejemplo 3:  $\nabla^2 V = 4$ 
4 - Ejemplo 4:  $\nabla^2 V = x/y + y/x$ 
5 - Ejecutar todos los ejemplos

Seleccione una opción (0-5): 1
Ingrese el número de threads a usar: 8
Ingrese el número de divisiones en X (M): 500
Ingrese el número de divisiones en Y (N): 500
Ingrese el tamaño de bloque (recomendado: 25-100): 50
```

Resultados: Actividad 7

```
==> RESULTADOS CON TASKS ==>
Tiempo de ejecución: 690.122 segundos
Número de iteraciones: 575861
Número de threads utilizados: 8
Resultados exportados a actividad7/ejemplo_0_tasks.dat
```

Ejemplo 0

```
==> RESULTADOS CON TASKS ==>
Tiempo de ejecución: 88.663 segundos
Número de iteraciones: 70234
Número de threads utilizados: 8
Resultados exportados a actividad7/ejemplo_1_tasks.dat
```

Ejemplo 1

```
==> RESULTADOS CON TASKS ==>
Tiempo de ejecución: 59.069 segundos
Número de iteraciones: 46706
Número de threads utilizados: 8
Resultados exportados a actividad7/ejemplo_2_tasks.dat
```

Ejemplo 2

Resultados: Actividad 7

Ejemplo 3

```
==> RESULTADOS CON TASKS ==>
Tiempo de ejecución: 46.567 segundos
Número de iteraciones: 39040
Número de threads utilizados: 8
Resultados exportados a actividad7/ejemplo_3_tasks.dat
```

Ejemplo 4

```
==> RESULTADOS CON TASKS ==>
Tiempo de ejecución: 85.72 segundos
Número de iteraciones: 67830
Número de threads utilizados: 8
Resultados exportados a actividad7/ejemplo_4_tasks.dat
```

20 hilos

Actividad 7: Exploración con tareas (task)

En esta actividad te proponemos experimentar con la directiva task de OpenMP para dividir el dominio en bloques y asignar tareas a distintos hilos.

```
== ACTIVIDAD 7: ECUACIÓN DE POISSON 2D CON TASKS ==

Ejemplos disponibles:
0 - Ejemplo Original: Término fuente gaussiano
1 - Ejemplo 1:  $\nabla^2 V = (x^2 + y^2)e^{xy}$ 
2 - Ejemplo 2:  $\nabla^2 V = 0$  (Ecuación de Laplace)
3 - Ejemplo 3:  $\nabla^2 V = 4$ 
4 - Ejemplo 4:  $\nabla^2 V = x/y + y/x$ 
5 - Ejecutar todos los ejemplos

Seleccione una opción (0-5): 1
Ingrese el número de threads a usar: 20
Ingrese el número de divisiones en X (M): 500
Ingrese el número de divisiones en Y (N): 500
Ingrese el tamaño de bloque (recomendado: 25-100): 50
```

Resultados: Actividad 7

Ejemplo 0

```
==> RESULTADOS CON TASKS ==>
Tiempo de ejecución: 395.233 segundos
Número de iteraciones: 577125
Número de threads utilizados: 20
Resultados exportados a actividad7/ejemplo_0_tasks.dat
```

Ejemplo 1

```
==> RESULTADOS CON TASKS ==>
Tiempo de ejecución: 55.39 segundos
Número de iteraciones: 73035
Número de threads utilizados: 20
Resultados exportados a actividad7/ejemplo_1_tasks.dat
```

Ejemplo 2

```
==> RESULTADOS CON TASKS ==>
Tiempo de ejecución: 33.617 segundos
Número de iteraciones: 49022
Número de threads utilizados: 20
Resultados exportados a actividad7/ejemplo_2_tasks.dat
```

Resultados: Actividad 7

```
==== RESULTADOS CON TASKS ===  
Tiempo de ejecución: 29.498 segundos  
Número de iteraciones: 39389  
Número de threads utilizados: 20  
Resultados exportados a actividad7/ejemplo_3_tasks.dat
```

Ejemplo 3

```
==== RESULTADOS CON TASKS ===  
Tiempo de ejecución: 47.515 segundos  
Número de iteraciones: 69297  
Número de threads utilizados: 20  
Resultados exportados a actividad7/ejemplo_4_tasks.dat
```

Ejemplo 4

32 hilos

Actividad 7: Exploración con tareas (task)

En esta actividad te proponemos experimentar con la directiva task de OpenMP para dividir el dominio en bloques y asignar tareas a distintos hilos.

```
==== ACTIVIDAD 7: ECUACIÓN DE POISSON 2D CON TASKS ===

Ejemplos disponibles:
0 - Ejemplo Original: Término fuente gaussiano
1 - Ejemplo 1:  $\nabla^2 V = (x^2 + y^2)e^{xy}$ 
2 - Ejemplo 2:  $\nabla^2 V = 0$  (Ecuación de Laplace)
3 - Ejemplo 3:  $\nabla^2 V = 4$ 
4 - Ejemplo 4:  $\nabla^2 V = x/y + y/x$ 
5 - Ejecutar todos los ejemplos

Seleccione una opción (0-5): 1
Ingrese el número de threads a usar: 32
Ingrese el número de divisiones en X (M): 500
Ingrese el número de divisiones en Y (N): 500
Ingrese el tamaño de bloque (recomendado: 25-100): 50
```

Resultados: Actividad 7

```
==> RESULTADOS CON TASKS ==>
Tiempo de ejecución: 396.359 segundos
Número de iteraciones: 577646
Número de threads utilizados: 32
Resultados exportados a actividad7/ejemplo_0_tasks.dat
```

Ejemplo 0

```
==> RESULTADOS CON TASKS ==>
Tiempo de ejecución: 50.086 segundos
Número de iteraciones: 73541
Número de threads utilizados: 32
Resultados exportados a actividad7/ejemplo_1_tasks.dat
```

Ejemplo 1

```
==> RESULTADOS CON TASKS ==>
Tiempo de ejecución: 33.982 segundos
Número de iteraciones: 49616
Número de threads utilizados: 32
Resultados exportados a actividad7/ejemplo_2_tasks.dat
```

Ejemplo 2

Resultados: Actividad 7

```
==== RESULTADOS CON TASKS ====
Tiempo de ejecución: 27.291 segundos
Número de iteraciones: 40254
Número de threads utilizados: 32
Resultados exportados a actividad7/ejemplo_3_tasks.dat
```

Ejemplo 3

```
==== RESULTADOS CON TASKS ====
Tiempo de ejecución: 48.016 segundos
Número de iteraciones: 70448
Número de threads utilizados: 32
Resultados exportados a actividad7/ejemplo_4_tasks.dat
```

Ejemplo 4

Conclusiones

¿Cómo se compara el rendimiento usando task con respecto a parallel for?

¿Qué ventajas y desventajas tiene este enfoque?

Cuadro 1: Comparación de Rendimiento por Número de Hilos

Ejemplo	Tiempo de Ejecución (segundos)			Mejora (%)	
	8 hilos	20 hilos	32 hilos	8→20 hilos	20→32 hilos
0	690.122	395.233	396.359	42.7 %	-0.3 %
1	88.663	55.390	50.086	37.5 %	9.6 %
2	59.069	33.617	33.982	43.1 %	-1.1 %
3	46.567	29.498	27.291	36.7 %	7.5 %
4	85.720	47.515	48.016	44.6 %	-1.1 %
Promedio	194.03	112.25	111.15	40.9 %	2.9 %

Conclusiones

¿Cómo se compara el rendimiento usando task con respecto a parallel for? ¿Qué ventajas y desventajas tiene este enfoque?

Cuándo usar Tasks

Problemas con carga computacional irregular

- Geometrías complejas o adaptativas
- Cuando se necesita balanceo de carga dinámico
- Algoritmos recursivos o con dependencias complejas

Cuándo usar Parallel For

- Mallas regulares con carga uniforme
- Cuando se requiere máximo rendimiento con mínimo overhead
- Algoritmos simples sin dependencias complejas
- Cuando la simplicidad del código es prioritaria

Conclusión

El enfoque de tasks ofrece mayor flexibilidad y potencial para balanceo de carga dinámico, pero a costa de mayor overhead y complejidad. Su efectividad depende fuertemente de:

- La regularidad del problema
- El tamaño de bloque elegido
- El número de hilos disponibles
- La arquitectura del sistema

Para el problema de Poisson 2D con malla regular, parallel for probablemente sería más eficiente, mientras que tasks brillaría en casos con geometrías irregulares o cargas computacionales variables.