Parcial No. 2

Paralelización del Juego del 15 (15-puzzle)

Parcial No. 2
Programación Paralela
Universidad Javeriana Cali
Jefferson A. Peña Torres

A	D	E	G
B	C	F	H
I	J	K	L
#	M	N	O

Estado inicial

A	D	E	G
B	C	F	H
#	J	K	L
I	M	N	O

Movimiento Arriba

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	#

Estado final

Descripción general

A continuación se presentan las directrices para lograr paralelizar uno o varios algoritmos secuenciales. Esta actividad evalúa el dominio de los conceptos básicos, las definiciones relacionadas con la implementación de hilos y/o múltiples procesos que permitan la ejecución simultánea de tareas y la implementación de un agente en C++.

Objetivos

Durante la actividad, los estudiantes lograrán lo siguiente:

- Identificar diferentes aspectos de la programación paralela
- Analizar un problema y descomponerlo (de ser posible) para generar una solución paralela al problema planteado.
- Comparar diferentes formas de programar en paralelo y determinar cuál solución ofrece las mejores ventajas

Contexto (Opcional)

El juego del 15 se implementa con un tablero 4x4 representado como una matriz bidimensional, donde 15 casillas contienen valores numerados y una permanece vacía. El objetivo es reorganizar los elementos mediante intercambios con la posición vacía hasta lograr el orden ascendente de izquierda a derecha y de arriba abajo. Entre 1880 y 1882, fue popular en los Estados Unidos y Europa. Sin embargo en 1882, se descubrió que de todos los problemas propuestos en el juego, sólo la mitad tenían solución.

Antes de comenzar

Revise, ejecute y ponga a prueba los fragmentos de código que se encuentran adjuntos a este documento.

Actividad 1: Entendimiento del problema

A continuación se describen las tareas a realizar para lograr la comprensión del desafío de programación.

(Tarea No. 1) El tablero: Escribe un programa en C++ que lea una cadena de texto, almacene la información en una estructura de datos apropiada e imprima el tablero como un cuadrado de 4×4.

- **Entrada:** Una cadena de caracteres y el símbolo #, que representa el espacio vacío. La posición de los caracteres en la cadena representa su ubicación en el tablero (de izquierda a derecha y de arriba hacia abajo). La cadena siempre tiene 16 caracteres (incluyendo el #), no tiene espacios, saltos de línea y letras en mayúsculas. Ejemplo:

```
Shell
# entrada
LAIB#KGCDOHENMJF
```

- **Salida:**Tablero como una matriz 4×4, es decir, 4 líneas y cada línea contiene 4 símbolos separados por espacio. Ejemplo:

```
Shell
# salida
L A I B
# K G C
D O H E
N M J F
```

(Tarea No. 2) Los movimientos: Escribe un programa en C++ que reciba una configuración inicial y una acción, y devuelva la configuración resultante

- **Entrada:** Dos líneas:
  - La cadena que representa el estado.
  - La acción: `UP`, `DOWN`, `LEFT` o `RIGHT`.
- **Salida:** La nueva configuración en formato 4×4.

```
Shell
# entrada:
LAIB#KGCDOHENMJF
RIGHT

# salida
L A I B
K # G C
```

```
D O H E
N M J F
```

Las acciones válidas son UP, DOWN, LEFT, RIGHT (siempre en mayúsculas). Si la acción no es posible desde el estado dado, imprimir sin cambios.

(Tarea No. 3) Acciones disponibles: Escribe un programa en C++ que determine qué acciones son posibles desde un estado dado.

- **Entrada:** Una cadena de caracteres y #.
- **Salida:** Lista de acciones válidas, una por línea, en el orden: `UP`, `DOWN`, `LEFT`, `RIGHT`. Solo imprimir las acciones que sean aplicables.

```shell
# entrada
ELFIGHJONAKDMB#C

# salida
UP
LEFT
RIGHT
```

Actividad 2: Estrategias seriales/secuenciales

A continuación se describen las tareas para realizar la implementación de forma secuencial

(Tarea No. 4) Búsqueda en anchura (Breadth-First Search): Implementa un programa en C++ que, dado un estado inicial, calcule el costo del camino más corto hasta el estado objetivo ABCDEFGHIJKLMNO#, suponiendo que cada acción cuesta 1.

- **Entrada:** Una línea con el estado inicial.
- **Salida:** El costo del plan óptimo.

```shell
# entrada
#AGCEBFDIJKHMNOL

# salida
8
```

Si no existe solución, imprimir la palabra "UNSOLVABLE" (o "-1", especificar cuál) en lugar de un número.

(Tarea No. 5) A* con distancia de fichas: Implementa el algoritmo A* usando la heurística *h1*, definida como el número de fichas que no están en su posición correcta.

- **Entrada:** Una línea con el estado inicial.
- **Salida:** Costo del camino óptimo al estado ABCDEFGHIJKLMNO#.

```
Shell
# entrada
EC#DBAJHIGFLMNKO

# salida
14
```

h1: número de fichas que no están en su posición final (no cuenta '#').

(Tarea No. 6) A* con distancia de fichas: Implementa el algoritmo A* usando la heurística *h2*, definida como la suma de las distancias de cada ficha hasta su posición final.

- **Entrada:** Una línea con el estado inicial.
- **Salida:** Costo del plan óptimo.

```
Shell
# entrada
EABCM#GDKFILNOJH

# salida
30
```

h2: suma de distancias Manhattan de cada ficha hasta su posición objetivo (distancia en filas+columnas).
El costo (número entero) es el plan óptimo, el número de acciones hasta ABCDEFGHIJKLMNO#.

(Tarea No. 7) Incremento del espacio de búsqueda: Modifique el problema para que admita tablero con más las letras del abecedario o para que en lugar de letras use números y permite cambiar el tamaño del tablero a 8x8, 16x16, 32x32, etc.

Actividad 3: Estrategia de paralelización

(Tarea No. 8) Descomposición de datos: Implementa una variante que divida el espacio en versiones más pequeñas y manejables para mejorar el análisis y procesamiento. Utilice hilos o procesos para lograr esta implementación. Utilice *std::thread*, OpenMP, MPI, procesos o POSIX threads para la implementación

(Tarea No. 9) Selección de la estrategia de descomposición: Implementa una variante en la que utilice descomposición de dominio, funcional o híbrida para resolver este problema. Además, describa la metodología utilizada para paralelizar el/los algoritmos.

- Métricas obligatorias: tiempo de ejecución (segundos), speedup (secuencial/parallel), eficiencia (speedup / n_hilos), y número de nodos expandidos por hilo.
- Incluir al menos 3 experimentos con distinto número de hilos/procesos.
- Dar formato: tablas y gráficos obligatorios en el informe.

Actividad 4: Evaluación y comparación

(Tarea No. 10) Comparación de algoritmos secuenciales: Usa los tres algoritmos (BFS, A*-h1, A*-h2) para resolver el conjunto de tableros que se encuentran entre los anexos (*puzzles.txt*) y evalúe:

- Cuántos nodos se expanden (cada vez que agregamos uno a la frontera).
- Registre la longitud de la solución y el número de nodos expandidos.

(Tarea No. 11) Comparación de versión paralelizada: Usa las variantes paralelizadas para resolver el conjunto de tableros que se encuentran entre los anexos y presente una evaluación comparativa entre ellas y las versiones secuenciales
(Tarea No. 12) Comportamiento al incrementar el espacio de búsqueda: Describa el comportamiento en la medida que incrementa el espacio de búsqueda (cambia el tablero de tamaño).

Entregable

- Informe de resultados con el análisis comparativo de los algoritmos
- Video corto sobre la implementación de cada tarea, los hallazgos y las comparaciones
- Incluya gráficos y tablas que permitan describir sus hallazgos

Letra pequeña

- La entrega debe realizarse en las fechas establecidas a través del aula digital. No se recibirán trabajos a través de correo electrónico.
- Agregue al informe los enlaces al código, al video o a cualquier otro recurso que haga parte de la entrega
- La longitud del video debe ser menor a 15 mins.
- Tenga en cuenta los criterios de evaluación descritos en la Rúbrica.
- Solo se reciben archivos en formato PDF a través de la plataforma.
- La implementación de cada tarea/]Actividad debe realizarse en C++

Tarea / Criterio	1 — Deficiente	2 — Insuficiente	3 — Aceptable	4 — Bueno	5 — Excelente	
T1. Representación del tablero (entrada/salida)	No compila o imprime formato incorrecto. No respeta tamaño ni posición del #.	Representa parcialmente el tablero; formato inconsistente o sin validación.	Imprime tablero 4×4 correcto, pero con pequeños errores de formato o validación.	Implementa correctamente la lectura y muestra formato limpio y ordenado.	Cumple 100% el formato, valida entradas y maneja errores con mensajes claros.	
T2. Movimiento de fichas (UP/DOWN/LEFT/RIGHT)	No ejecuta movimientos.	Ejecuta algunos movimientos pero con errores de posición.	Ejecuta todos los movimientos válidos; falla ante casos inválidos.	Ejecuta correctamente movimientos válidos e ignora los no posibles.	Implementa movimientos correctos, manejo de límites y formato impecable.	
T3. Acciones disponibles	No determina acciones válidas.	Lista incompleta o en orden incorrecto.	Lista acciones válidas pero sin respetar orden requerido.	Lista acciones válidas y en orden correcto.	Lista exacta, en orden correcto, y con validación perfecta del estado.	
T4. Búsqueda en Anchura (BFS)	No compila o no encuentra camino.	BFS incompleto o incorrecto (no garantiza camino mínimo).	BFS funcional pero sin manejo de estados repetidos o sin control de memoria.	BFS correcto, con control básico de repetidos y salida válida.	BFS óptimo, con control de estados, salida correcta y tiempos razonables.	
T5. A* con h1 (fichas fuera de lugar)	No ejecuta A*.	Implementa búsqueda pero sin heurística o incorrecta.	Implementa A* con h1 pero con errores de conteo o expansión.	Implementa A* con h1 correctamente y obtiene resultados válidos.	A* con h1 completamente funcional, eficiente y validado contra BFS.	
T6. A* con h2 (distancia Manhattan)	No ejecuta o confunde heurística.	Implementa heurística incorrecta.	Implementa Manhattan pero sin excluir # o sin expansión óptima.	Implementa A* con h2 correcto, salida coherente.	Implementa A* con h2 óptimo, eficiente y validado.	
T7. Extensión a tableros mayores (8×8, 16×16...)	No modifica código o falla.	Permite cambio parcial de tamaño, no generaliza.	Permite tamaños mayores, pero requiere ajustes manuales.	Generaliza tamaño correctamente, con lectura flexible.	Permite tamaño variable y tipo de ficha (letras/números) con validación dinámica.	
T8. Descomposición de datos (paralelo)	No implementa paralelismo.	Paralelismo incorrecto o sin efecto (ejecución secuencial).	Usa hilos pero sin sincronización ni medición.	Divide datos y usa hilos/procesos con resultados correctos.	Paraleliza correctamente con sincronización, medición y mejora demostrable.	
T9. Estrategia de descomposición (dominio/funcional/híbrida)	No describe ni implementa estrategia.	Implementa parcialmente o sin justificación.	Describe y aplica una estrategia, sin medir rendimiento.	Aplica estrategia coherente y reporta rendimiento.	Implementa, justifica y compara al menos dos estrategias con métricas claras.	
T10. Comparación de algoritmos secuenciales	Sin comparación ni resultados.	Resultados incompletos o sin formato.	Compara parcialmente (solo tiempo o nodos).	Compara BFS, A*-h1, A*-h2 con métricas correctas.	Compara y analiza exhaustivamente con tablas y gráficos claros.	
T11. Comparación de versiones paralelas vs. secuenciales	No ejecuta comparaciones.	Comparación superficial o sin datos.	Reporta tiempos pero sin análisis.	Analiza tiempos y nodos, con interpretación básica.	Presenta comparación completa: speedup, eficiencia, gráficas y conclusiones.	
T12. Comportamiento al aumentar espacio de búsqueda	No analiza cambio de tamaño.	Muestra resultados sin interpretación.	Analiza parcialmente los efectos del tamaño.	Analiza claramente el impacto en tiempo y nodos.	Analiza y discute escalabilidad con argumentos técnicos y evidencia empírica.	
Informe escrito (formato y contenido)	No entregado o incompleto.	Desorganizado, sin análisis ni evidencias.	Presenta estructura mínima con resultados parciales.	Bien organizado, con resultados y conclusiones.	Informe profesional: completo, visualmente claro, con discusión y referencias.	
Video de presentación (<10 min)	No entregado.	Video fuera de tiempo o sin explicación clara.	Muestra código pero sin interpretación de resultados.	Explica implementación y resultados con claridad.	Video conciso, técnico y bien narrado que demuestra dominio total.	