UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA

ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS

PROYECTO - Resolución Paralelizada del Rompecabezas Futoshiki en CUDA

ASIGNATURA: **ALGORITMOS PARALELOS Y DISTRIBUIDOS**

DOCENTE: **HANS HARLEY CCACYAHUILLCA BEJAR**

ESTUDIANTE: **Daniel Montañez Medina**

CÓDIGO: **210936**

CUSCO – PERÚ

2024

Informe: Resolución Paralelizada del Rompecabezas Futoshiki en CUDA

Introducción

El algoritmo desarrollado en CUDA busca resolver el rompecabezas de Futoshiki, un desafío lógico en el que se deben llenar celdas con números del 1 al 5, cumpliendo condiciones de unicidad en filas y columnas, así como restricciones de desigualdad entre ciertas celdas.

Estructura y Paralelización del Código

La implementación se organiza en tres secciones claves que aprovechan la arquitectura paralela de CUDA:

1. Almacenamiento de Restricciones en Memoria Constante

```
__constant__ char dev_restricciones_fila[N][N];
__constant__ char dev_restricciones_columna[N][N];
```

Las restricciones de filas y columnas se almacenan en la memoria constante de CUDA, optimizando el acceso a estas condiciones y permitiendo que múltiples hilos las consulten simultáneamente sin afectar el rendimiento.

2. Verificación de Validez en Paralelo

```
__device__ bool es_valido(int tablero[N][N], int fila, int columna, int num) {
    // Validación de unicidad y desigualdad
}
```

Función: Esta función comprueba si un número puede colocarse en una celda específica, garantizando que cumple con las condiciones del juego (unicidad en filas/columnas y restricciones de desigualdad). Cada hilo realiza esta verificación, evaluando en paralelo múltiples posibles valores y posiciones.

3. Kernel de Resolución con Backtracking

```
__global__ void resolver_futoshiki(int* solucion_encontrada, int tablero[N][N], int fila,
    // Implementación de backtracking en GPU
}
```

Este kernel implementa un backtracking en GPU, explorando las posibles soluciones de forma paralela. Usa atomicExch para marcar la primera solución válida encontrada, garantizando sincronización entre hilos. CUDA gestiona la resolución en diferentes hilos y niveles de profundidad, acelerando la búsqueda de soluciones.

4. Transferencia de Datos entre CPU y GPU

```
cudaMemcpyToSymbol(dev_restricciones_fila, h_restricciones_fila, N * N * sizeof(char));
```

Se copian las restricciones y el tablero inicial a la GPU, minimizando la comunicación entre host y dispositivo para optimizar el rendimiento general del algoritmo.

Conclusión

La paralelización en CUDA permite resolver el rompecabezas de Futoshiki de forma eficiente, aprovechando memoria constante, sincronización con operaciones atómicas y backtracking paralelo. Este enfoque reduce significativamente el tiempo de cálculo en comparación con un método secuencial en CPU, demostrando la efectividad de CUDA en resolver problemas de lógica complejos.