Juego de la Vida

1. Objetivos

La realización de esta práctica persigue los siguientes objetivos:

- Familiarizar al alumno con el modelo de programación paralela basado en paso de mensajes (MPI), utilizado en sistemas de memoria distribuida.
- Introducir los conceptos básicos de la programación paralela híbrida en sistemas compuestos por nodos un conjunto de cómputo con múltiples cores cada uno.
- Diseñar e implementar un algoritmo de equilibrio de carga de trabajo para optimizar el rendimiento de las aplicaciones en sistemas heterogéneos.
- Analizar el rendimiento y la escalabilidad de la aplicación propuesta, tanto en cuanto al escalado fuerte como al débil.

2. Descripción del problema

El juego de la vida es un autómata celular inventado por el matemático de Cambridge John Conway. Consiste en un tablero compuesto por un conjunto de celdas. Iterativamente se van recorriendo las celdas y en función de unas sencillas reglas matemáticas cada celda puede vivir, morir o multiplicarse. Dependiendo de las condiciones iniciales, las celdas forman una serie de patrones en el curso del juego.

En esta versión del juego tenemos un tablero cuadrado (2D) donde las celdas se actualizan a cada paso del tiempo de acuerdo con las siguientes reglas:

- Si una celda está ocupada:
  - Celdas con 0 o 1 vecinos, mueren, por soledad.
  - Celdas con 4 o más vecinos, muere, por sobrepoblación.
  - Celdas con 2 ó 3 vecinos, sobreviven.
- Para una celda vacía, es decir no poblada:
  - Cada celda con 3 vecinos se convierte en ocupada.

En el resto de los casos las casillas quedan como estaban en la iteración anterior.

3. Material Necesario

Se proporciona un único archivo `life.zip`, que incluye dos ficheros:

- *life.c*: Contiene el código necesario para implementar el juego de la vida, con la reglas descritas en el apartado anterior.
- *life.in*: Contiene un ejemplo sencillo con un tablero de 11×11 casillas para poder visualizar los pasos que se ejecutan.
- *judge.in*: Fichero con un tablero de 500×500, que servirá para realizar las pruebas de rendimiento.

El programa recibe tres parámetros de entrada que se proporcionan por línea de comandos, como en el siguiente ejemplo:

```
time ./life life.in 11 7
```

- Nombre del fichero que contiene la inicialización del tablero, es decir las casillas que están habitadas inicialmente (life.in, en este ejemplo).
- Número tamaño del tablero (sólo una dimensión). En el ejemplo tenemos un tablero de 11×11
- Número de pasos o jugadas a desarrollar en cada ejecución. En el ejemplo se ejecutan solamente 7 pasos.

Si no se proporcionan los datos correctos el programa puede fallar, produciendo un error de acceso a memoria.

La salida, para el tablero pequeño, se puede observar en la pantalla descimentado la línea del código `#define DEBUG 1`. Para tableros más grandes, no caben en la pantalla y no se puede observar la evolución.

Para realizar las pruebas de depuración y ver que los resultados de la paralelización son correctos se usará el fichero `life.in`, con los parámetros indicados. Para realizar las pruebas de evaluación de rendimiento y escalado se utilizará el fichero `judge.in` con el tamaño 500 y un número de pasos que obtengan un tiempo de ejecución de entre 10 y 20 segundos, según el ordenador empleado para hacer la práctica. [1]

4. Desarrollo

Ejercicio 1

Realiza la mejor paralelización del código posible utilizando el modelo de programación paralela de paso de mensajes, MPI. Trabajaremos sobre un sistema compuesto por cuatro nodos de cómputo, por lo que será necesario lanzar cuatro procesos. Comprueba que el

[1] En mi equipo se consigue con 1500 pasos.

resultado del programa sea correcto, comparando los resultados con los obtenidos en el programa secuencia. Mide el tiempo de ejecución del programa secuencial y del paralelo. Calcula el speedup y la eficiencia obtenida y explica los resultados.

Ejercicio 2

Consideramos ahora que cada uno de los nodos de cómputo sobre los que trabajamos es un sistema compuesto por dos procesadores, con multi-threading. Para explotar esta arquitectura es necesario utilizar programación paralela híbrida, mezclando los modelos de paso de mensajes y memoria compartida. Por lo tanto, es necesario paralelizar cada uno de los procesos MPI obtenidos en el ejercicio anterior con directivas de OpenMP. Realiza la mejor paralelización posible, añadiendo al código las directivas y/o funciones de OpenMP necesarias. Etiqueta todas las variables y explica por qué le has asignado esa etiqueta. Calcula el speedup y la eficiencia obtenida y explica los resultados.

Ejercicio 3

Implementar un algoritmo de equilibrio de carga de trabajo que sea capaz de distribuir los datos entre los procesos, de forma que cada uno tenga una cantidad de trabajo proporcinal a su potencia de cómputo. El objetivo de este algoritmo es que todos los procesos finalicen su ejecución aproximadamente a la vez. El algoritmo será centralizado y debe tener en cuenta la heterogeneidad del sistema.

Ejercicio 4

Determina, hasta dónde sea posible con el computador con el que estás trabajando, el escalado fuerte y el escalado débil de esta aplicación. Explica las pruebas que has realizado para ello. Extrae todas las conclusiones que puedas sobre el comportamiento de este programa.

5. Evaluación

La evaluación de esta práctica se realizará medinate una presentación del trabajo realizado en el aula, el día que se indique en el Moodle de la asignatura. La duración de la presentación será de 10 minutos. Además se deberá entregar dicha presentacion, así como el código realizado, mediante una tarea de la plataforma Moodle.