Práctica 3
Paralelismo a nivel de hilos
Parte II: Paralelización mediante OpenMP de algoritmos neuroevolutivos

Objetivos:

- Aprender a paralelizar una aplicación en una máquina paralela de memoria centralizada mediante hilos (*threads*) usando el estilo de *variables compartidas*.
- Estudiar el API de OpenMP y aplicar distintas estrategias de paralelismo en su aplicación.
- Aplicar métodos y técnicas propios de esta asignatura para estimar las ganancias máximas y la eficiencia del proceso de paralelización.
- Aplicar todo lo anterior a un problema de complejidad y envergadura suficiente.

Desarrollo:
En esta práctica los estudiantes deben paralelizar, usando OpenMP, la solución a un problema dado para aprovechar los distintos núcleos de los que dispone cada ordenador de prácticas. Se paralelizará por tanto para un sistema multiprocesador (máquina paralela de memoria centralizada), en el que todos los núcleos de un mismo encapsulado ven la misma memoria, es decir, un puntero en un núcleo es el mismo puntero para el resto de los núcleos del microprocesador.

La práctica está dividida en 2 partes. La primera, está pensada para que los alumnos se familiaricen con OpenMP y las posibilidades que ofrece para paralelizar soluciones a problemas que se puedan utilizar en sistemas multiprocesador. En la segunda parte se propondrá un problema concreto cuya solución y estudio se utilizará lo aprendido en la primera parte.

Tarea 3: Paralelización de algoritmos neuroevolutivos mediante OpenMP

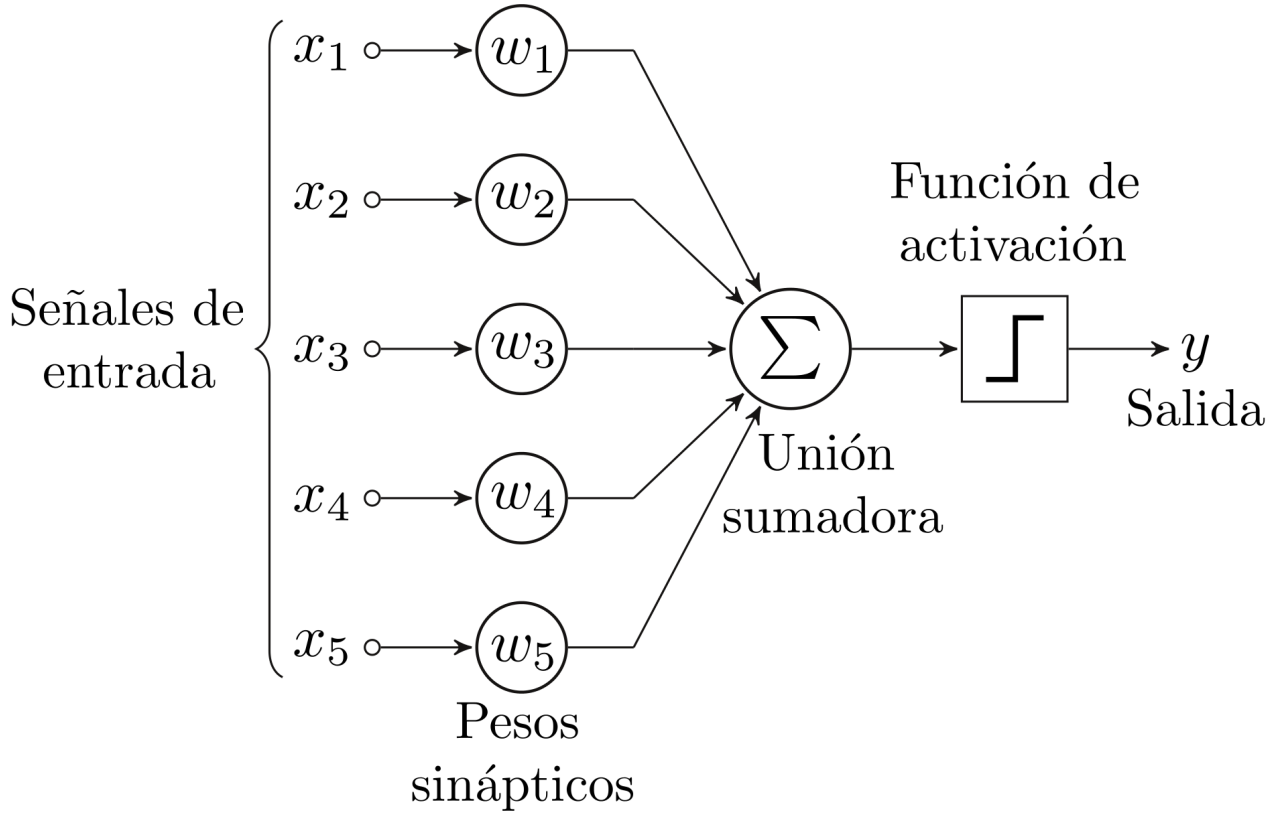
Todos los grupos de cada turno de prácticas deben acometer la paralelización del problema planteado. Para ello se analizará la solución secuencial facilitada siguiendo las indicaciones del enunciado y los profesores. Posteriormente, se transformará ésta, utilizando OpenMP, para que incorpore paralelismo a nivel de hilos.

Problema:
El problema planteado es la optimización de un algoritmo neuroevolutivo cuyo objetivo es generar el mejor individuo capaz de moverse por un espacio 2D pasando por puntos concretos del espacio.

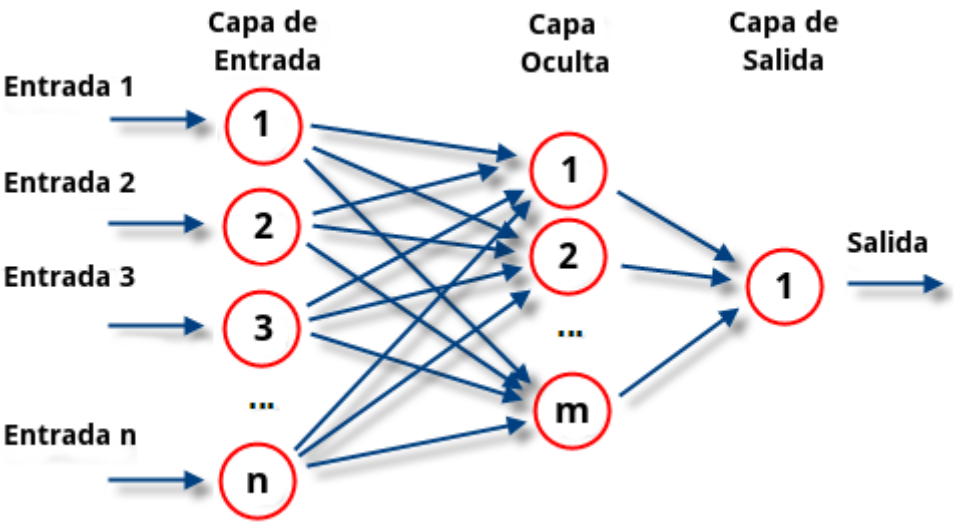
Un algoritmo neuroevolutivo combina dos técnicas de machine learning, por un lado, utiliza redes neuronales artificiales que se caracterizan por ser aproximadores de funciones universales; por otro lado, utiliza un algoritmo genético para entrenar la red neuronal y obtener la función deseada.

Red neuronal artificial

Una red neuronal artificial simula las interacciones entre las neuronas de un cerebro biológico. El equivalente a una neurona biológica es el perceptrón. El perceptrón es la unidad más simple que forma una red neuronal artificial. Éste tiene un conjunto de entradas a las cuales aplica el producto escalar con un vector de pesos individuales, es decir, una combinación lineal de los valores de entrada. Finalmente, aplica una función de activación no lineal para dar un único valor de salida. La finalidad de esta función de activación es poder aproximar funciones no lineales.



La salida de un perceptrón puede ser la entrada de otro perceptrón, formando así una red de neuronas artificiales conocida como perceptrón multicapa. Por lo general, cada perceptrón puede tener un número arbitrario de entradas, sin embargo, para facilitar la computación, los perceptrones se suelen agrupar por capas donde en cada capa, los perceptrones tienen tantas entradas como número de perceptrones hay en la capa anterior, es decir, cada perceptrón de la capa n tiene como entrada las salidas de cada perceptrón de la capa n-1. Por tanto, habrá una capa inicial de entrada, m capas intermedias (u ocultas) y una capa de salida que proporcionan la aproximación a la función y = F(x) donde y es el vector formado por los valores de las salidas de los perceptrones que forman la capa de salida, x un vector formado por las entradas a la red y F(x) la función generada por la red.



La forma tradicional de conseguir una F(x) determinada utiliza el algoritmo de backpropagation para ajustar los pesos de la red dependiendo del error cometido al evaluar un conjunto de pares (x, y) conocidos. Los pesos finales se habrán modificado para minimizar la diferencia entre la salida deseada y la salida que devuelve la red.
El principal inconveniente de esta forma de entrenar una red es que necesitas disponer de un conjunto amplio de pares (x, y). Además, esos pares deben estar distribuidos lo más uniformemente posible para que la función resultante sea lo más cercana a la deseada. Un forma de entrenar la red si no se dispone de esos pares (x, y) es utilizar otros algoritmos de optimización como un algoritmo genético.

Algoritmo genético

Un algoritmo genético es una búsqueda heurística inspirada en la teoría de la evolución de Charles Darwin. Este algoritmo refleja el proceso de selección natural para seleccionar a los individuos más aptos de una población. Ellos producen descendientes que heredan las características de los padres y se añadirán a la siguiente generación. Si los padres tienen una mejor aptitud, su descendencia será mejor que los padres y tendrá una mejor oportunidad de sobrevivir. Este proceso sigue en iteración y al final, se encontrará una generación con los individuos más aptos. Se consideran cinco fases en un algoritmo genético.

1. **Población inicial.**
   Cada **individuo** de esta **población** es una potencial solución al problema y se caracteriza por un conjunto de parámetros (variables) conocidos como **genes**. Los genes se unen en una cadena para formar un **cromosoma** (solución).
2. **Estimación de la función de *fitting***
   Determina el grado de aptitud de un individuo (la capacidad de un individuo para competir con otros individuos). Dado un individuo, esta función puntúa su validez. La probabilidad de que un individuo sea seleccionado para la reproducción se basa en esta puntuación.
3. **Selección**
   En esta fase se escogen los individuos más aptos, según la función de *fitting*, para que sus genes pasen a la siguiente generación.
4. **Cruce**

Esta fase es la más significativa en un algoritmo genético. Para cada pareja de padres que se cruzan, se elige al azar un punto de cruce dentro de los genes. Se obtiene la descendencia mediante el intercambio de los genes de los padres entre ellos hasta que se alcanza el punto de cruce. Posteriormente se añade a la población.

5. **Mutación**
   Algunos de los genes de los nuevos individuos pueden ser sometidos a una mutación en función de una probabilidad aleatoria, generalmente baja. La mutación se produce para mantener la diversidad dentro de la población y evitar la convergencia temprana dejando sin explorar ciertas soluciones.

En nuestro caso, **el cromosoma serán los pesos de la red** cuyos valores para la generación inicial serán aleatorios entre -1 y 1. La **función de fitting** es un contador del número de checkpoints a los que el individuo ha logrado llegar. Y el **cruce** (*crossover*) es un bucle que recorre el cromosoma determinando si el peso debe ser el de un individuo o el otro o si debe mutar alguno de ellos. En cada generación, se debe calcular el cruce entre los N mejores individuos y calcular el *fitting* de cada uno de ellos. En todas esas operaciones se puede aplicar paralelismo.
La función de fitting se calcula mediante una simulación en la que se pone a prueba a cada individuo para ver si son capaces de moverse por un espacio 2D siguiendo una serie de puntos. Una vez finalizada la simulación, el valor de fitting de cada individuo será el número de puntos que ha logrado alcanzar.

Tarea 3.1: Analiza el código e identifica las fases del algoritmo genético
En esta tarea deberéis analizar el código utilizando las estrategias vistas en la práctica anterior para identificar las fases del algoritmo genético y elaborar un diagrama de flujo.
Identifica también si se puede ejecutar algunas fases en paralelo y las dependencias entre ellas.

Tarea 3.2: Analiza el código de la simulación
En esta tarea deberéis analizar el código correspondiente a la simulación que calcula la función de fitting. Realiza un diagrama de flujo de la simulación e identifica posibles puntos paralelizables.

Tarea 3.3: Analiza el código del MLP
De la misma forma que en las tareas anteriores, analiza el código del perceptrón multicapa, obtén el diagrama de flujo e identifica donde se puede aplicar paralelismo.

Tarea 3.4 Análisis global
Una vez que tenéis los 3 diagramas de flujo, analizad cómo interaccionan entre sí realizando un diagrama de flujo conjunto.

Tarea 3.5 Paralelización
Identificadas las partes paralelizables del código, ahora toca aplicar el paralelismo con OpenMP. Haced pruebas paralelizando diferentes partes y observad la ganancia de rendimiento.
Responded a las siguientes preguntas:

- ¿Se obtiene un mejor rendimiento paralelizando todos las partes posibles?
- ¿Se degrada el rendimiento al paralelizar ciertas partes?
- Si es así, ¿a qué creéis que se debe esa degradación?

Objetivos:

- Obtener una versión paralela con OpenMP de la solución secuencial proporcionada
- Analizar y explotar en todos los casos los diferentes tipos posibles de paralelismo que se perciban en la solución secuencial dada.
- Realizar pruebas suficientes que permitan estimar valores de rendimiento de la versión paralela frente a la secuencial.

Pasos:

a. Analizar la solución secuencial facilitada para asegurar la comprensión del problema. Prestar atención a detalles vistos en prácticas anteriores que tengan importancia para diseñar la solución paralela: variables, talla del problema, etc.
b. Realice una representación gráfica en forma de grafo de control de flujo del funcionamiento de la solución secuencial dada.
c. Utilizando OpenMP y a la vista de los detalles anteriores, diseñe una solución paralela a nivel de hilo a partir de la versión secuencial
d. Realice pruebas de su versión paralela para diferentes casos y configuraciones tomando medidas de tiempos y ganancias.
e. Realice una representación gráfica en forma de grafo de control de flujo del funcionamiento de la solución paralela basada en OpenMP.
f. Comente los siguientes aspectos tanto de la solución secuencial como de la paralela según se indique:

Sobre el código implementado

- Comentar las porciones de código de más interés tanto de la solución secuencial como de la paralela implementadas:
  - Aspectos que definan la talla del problema.
  - Estructuras de control del código de especial interés en la solución al problema.
  - Es importante que se justifique **lo más detalladamente** posible los cambios que se hayan realizado con respecto a la versión secuencial para paralelizar la solución con OpenMP.
  - Instrucciones y bloques de OpenMp utilizados para la paralelización del código.

Sobre el paralelismo explotado

- **Revisando la documentación de la "*Unidad 3. Computación paralela*".** Indique lo siguiente con respecto a su práctica:
  - Tipos de paralelismo usado.
  - Modo de programación paralela.
  - Qué alternativas de comunicación (explícitas o implícitas emplea su programa).
  - Estilo de la programación paralela empleado.
  - Tipo de estructura paralela del programa

Sobre los resultados de las pruebas realizadas y su contexto

- Caracterización de la máquina paralela en la que se ejecuta el programa. (p.ej. Número de nodos de cómputo, sistema de caché, tipo de memoria, etc.).
  En Linux use la orden: `cat /proc/cpuinfo` para acceder a esta información.
- ¿Qué significa la palabra `ht` en la salida de la invocación de la orden anterior? ¿Aparece en su ordenador de prácticas?
- Calcular lo siguiente (con gráficas asociadas y su explicación breve):
  - Ganancia en velocidad (*speed-up*) en función del número de unidades de cómputo (*threads* en este caso) y en función de los parámetros que modifiquen el **tamaño del problema** (p.ej. dimensiones de una matriz, número de iteraciones considerado, etc.…).
    - Comente los resultados de forma razonada. ¿Cuál es la ganancia en velocidad máxima teórica?
      Nota: Puede entregar gráficas en 2D o 3D según resulte más ilustrativo.
  - **Responda de forma justificada:** ¿Cuál es la implementación más eficiente de las 2?

Tarea 4: Competición de los mejores individuos

Para evaluar la optimización obtenida por cada grupo, la última sesión de prácticas se hará una competición en la cual se enfrentará el mejor individuo entrenado por cada grupo tras un tiempo de entrenamiento de entre 5 y 10 min. Pasado ese tiempo, los grupos pararán el entrenamiento y entregarán al profesor el archivo con la última generación. El profesor cargará un fichero por cada grupo y se quedará con el mejor individuo de cada fichero para lanzar una simulación donde puedan competir. Se evaluará tanto el número de checkpoints que logre pasar como el tiempo empleado en caso de empate.

Tarea 5: Redacción de una memoria en la que se analice el diseño utilizando OpenMP y los resultados obtenidos.

Se redactará una memoria en la que se comenten y expliquen diferentes aspectos tanto de la versión secuencial como de la implementación paralela propuesta. También se analizarán los resultados obtenidos probándolas. La memoria deberá dar clara respuesta a las cuestiones planteadas en cada tarea. Cada profesor indicará la forma en que les será entregadas las partes individuales de cada miembro de los grupos.

Notas generales a la práctica:

- Las respuestas y la documentación generadas tanto en las tareas de la parte I como de la II se integrarán en la memoria conjunta y estructurada, incluyendo los apartados individuales como anexo, que se entregará al final de la práctica 3.
- Para el trabajo en la parte II y en la memoria se utilizará la plataforma de desarrollo GitHub. Esta permitirá una gestión de un **repositorio común de trabajo** que deberá crear cada grupo incluyendo a sus miembros y al profesor. Este dará a conocer su usuario en dicha plataforma para ser añadido.

- La implementación realizada tendrá que poder ejecutarse bajo el sistema operativo Linux del laboratorio de prácticas, aunque en casa puede trabajar con otros compiladores y sistemas operativos que soporten OpenMP (icc, clang, Visual C++, ...).
- Para compilar, el código adjuntado incluye un archivo CMakeLists.txt que contiene la información de dependencias de librerías externas y cómo construir los fuentes. Este archivo sirve para construir un Makefile utilizando CMake. Para ello, crea una carpeta llamada build en el mismo directorio donde se encuentra la carpeta src y ejecuta el comando `cmake -S . -B ./build`. Una vez ejecutado el comando, dentro de la carpeta build habrá un fichero Makefile con el que poder ejecutar el comando make para construir el ejecutable.
- Para ejecutar la simulación del algoritmo neuroevolutivo hay que usar el comando `./evolution follow [-s]`. El argumento -s se usa para poder visualizar la simulación de todas las generaciones, por lo general, se ejecutará sin -s para evaluar el rendimiento por generación sin que impacte el tiempo de visualización.
- La información referente a OpenMP se encuentra en www.openmp.org. Para compilar use el make generado por cmake que ya incluye el flag -fopenmp.
- La práctica se deberá entregar mediante el método que escoja su profesor de prácticas antes de la sesión de prácticas de la semana del 28 de noviembre de 2022.

***Nota:**

Los trabajos teóricos/prácticos realizados han de ser originales. La detección de copia o plagio supondrá la calificación de “0” en la prueba correspondiente. Se informará la dirección de Departamento y de la EPS sobre esta incidencia. La reiteración en la conducta en esta u otra asignatura conllevará la notificación al vicerrectorado correspondiente de las faltas cometidas para que estudien el caso y sancionen según la legislación.