CCPD

Topic 1: Concurrency::CCPD

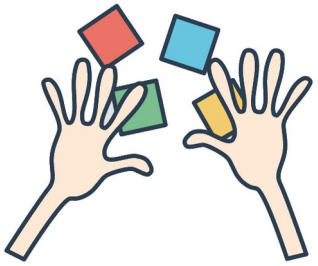
Lab 6: OpenMP

Week 6 of threads::CCPD
2023/2024

David Olivieri
Leandro Rodriguez Liñares
Uvigo, E.S. Informatica

Hands-on: Lab6prog01

Lecture Code



Hands-on: Lab6prog01

1. Crear y configurar el entorno Anaconda
   - Desde tu terminal o consola, ejecutar:

```
conda create -n mi_entorno gcc_linux-64 gxx_linux-64
```

   - Esto creará un nuevo entorno llamado `mi_entorno` con los compiladores de C/C++ (GCC) necesarios para trabajar con OpenMP.

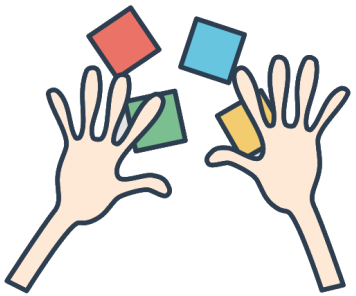
2. Leer, comprender y ejecutar los archivos “.c” y “.py” vistos en clase
   - Clonar o copiar los archivos fuente con ejemplos de OpenMP en C (`.c`) y Python con wrappers o ejemplos (`.py`).
   - Revisar el código para entender la estructura y las directivas de OpenMP (`#pragma omp parallel`, `#pragma omp for`, etc.).
   - Compilar los programas en C con la bandera `-fopenmp` (por ejemplo:

```
`gcc -fopenmp -o mi_programa mi_programa.c  -fopenmp`
```

     y ejecutarlos.
   - En Python, probar los scripts para verificar que estén funcionando correctamente.
   - Observar la salida y confirmar el paralelismo (por ejemplo, verificar que se reparten iteraciones entre hilos).

Hands-on:
Lab6prog02

K-Means Parallelo con OpenMP



Hands-on: Lab6prog02

1. Ejecutar el algoritmo de K-means en paralelo usando OpenMP
    - Implementar o adaptar el código de K-means para que use directivas OpenMP.
    - Si lo prefieres, puedes hacerlo en C (compilando con `-fopenmp`) o en Python (usando PyOMP o un wrapper que llame a funciones C con OpenMP).
    - Asegúrate de paralelizar, al menos, la fase de asignación de cada punto al centroide más cercano y/o la fase de actualización de centroides.

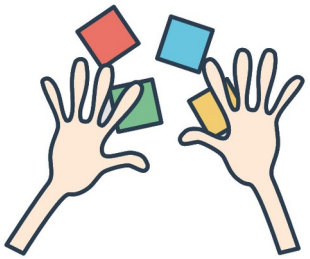
2. Utilizar datos reales para la versión paralela de K-means
    - En lugar de datos simulados (random), usar un pequeño conjunto de datos reales (por ejemplo, algún CSV con coordenadas 2D o 3D).
    - Preprocesar los datos si hace falta (leer el fichero, normalizar, etc.).
    - Ejecutar el programa y comprobar:
        - El número de iteraciones y los centroides finales.
        - El tiempo de ejecución (comparar paralelismo vs. versión secuencial si es posible).

Objetivo Final: Familiarizarse con el uso de OpenMP (o PyOMP) para acelerar un algoritmo típico de machine learning (K-means), observar mejoras de rendimiento y practicar con datos reales.

Hands-on:
Lab6prog03

K-means con Python + PyOMP

Optional



Hands-on: Lab6prog03 (optional)

En este ejercicio, en lugar de ejecutar el código `.c` directamente (como en el ejercicio 2), van a crear un “wrapper” para poder llamar esas mismas funciones desde Python:

1. Crear el módulo PyOMP:

- Escribir un archivo (por ejemplo, `kmeans_omp.c`) con las mismas funciones de K-means paralelizadas con `#pragma omp`.
- Preparar un `setup.py` (o equivalente) que:
  1. Compile `kmeans_omp.c` con `-fopenmp`.
  2. Genere un módulo Python importable, p. ej. `pyomp.so` o `pyomp.pyd`.
     - Instalar/compilar con:

```
python setup.py build_ext --inplace
```

- Asegurarse de tener el entorno con gcc/g++ (en Anaconda: `gcc_linux-64 gxx_linux-64`).

2. Importar y usar en Python:

- Desde un script de Python (p. ej. `kmeans_pyomp.py`), hacer:

```
import pyomp  # el módulo que crearon
```

-
- Llamar las funciones definidas en C (p. ej. `assign_points_omp()` y `accumulate_omp()`) pasando arrays de NumPy.

3. Ejecutar y comprobar:

- Ejecutar `python kmeans_pyomp.py` para correr K-means en paralelo.
- Comparar tiempos con la versión del ejercicio 2 (directo en C) o con una versión secuencial en Python.