IMT2112 Semestre 2025-2
Tarea 2

Elwin van 't Wout

27 de agosto de 2025

Introducción

El algoritmo PageRank es un método para calcular la importancia de nodos en un grafo. Este algoritmo ganó fama cuando era utilizado por Google en su buscador de web. En este caso, el web es representado por un grafo, con cada página un nodo y los hyperlinks las conexiones (aristas). A su vez, el grafo es representado por una matriz y, después de algunas manipulaciones, el vector propio principal entrega la importancia de las páginas. Una parte esencial y costosa en este algoritmo es encontrar el valor propio más grande de una matriz. El 'método de potencia' (*power iteration*) es un algoritmo iterativo que aproxima el vector propio más grande de una matriz.

El método de potencia funciona como siguiente. Elegimos un vector inicial $\mathbf{b}_0$ distinto a cero. Lo más común es elegir un vector con valores aleatorios o un vector con todos los elementos igual a uno. Dada una matriz $A \in \mathbb{R}^{n\times n}$ diagonalizable, la iteración

$$\mathbf{b}_{k+1} = \frac{A\mathbf{b}_k}{||A\mathbf{b}_k||}, \quad k = 0, 1, 2, \ldots$$

converge al vector propio de A que corresponde al valor propio más grande. La secuencia

$$\mu_k = \frac{\mathbf{b}_k^T A\mathbf{b}_k}{\mathbf{b}_k^T\mathbf{b}_k}$$

converge al valor propio más grande. Dado que el vector propio es normalizado en este algoritmo, tenemos $\mathbf{b}_k^T\mathbf{b}_k = 1$, lo cual simplifica la expresión general.

Tarea

Esta tarea contempla una implementación del método de potencia con MPI.

1. Corre el código de Python disponible en Canvas. Este código calcula una matriz diagonalizable con valores propios reales entre 1 y 10. Si quieren,

pueden cambiar el tamaño de la matriz con la variable `ndim`. La matriz es exportada a un archivo de texto. Los primeros dos números en el archivo son las dimensiones de la matriz. Después, cada línea en el archivo corresponde a un único elemento de la matriz, contado fila por fila. En Canvas, pueden encontrar un código de C++ que lea un archivo de texto.

2. El objetivo de la tarea es implementar el método de potencia en paralela con MPI. La variante de paralelización debe ser en base a la partición por bloques de filas. Explique, en palabras, cómo se puede paralelizar el método de potencia. Más precisa:

   a) ¿Se puede paralelizar la iteración principal sobre k?

   b) ¿Cómo se puede paralelizar la multiplicación matriz por vector?

   c) ¿Cómo se puede paralelizar el producto punto entre vectores?

3. Implemente el método de potencia en paralelo, haciendo las instrucciones siguientes. Deben usar C++ y MPI para todo lo que sigue.

4. Lee el tamaño de la matriz desde el archivo y calcule el tamaño de cada bloque de la matriz en cada proceso. Ojo que el código debe funcionar para cualquier número de procesos y dimensión de la matriz. En específico, también cuando la dimensión no es un múltiplo del número de procesos. Explique cómo exactamente calcularon el tamaño de cada bloque.

5. Inicialice el vector $\mathbf{b}_0$ con todos los componentes iguales a uno. Este vector también debe ser particionado sobre los procesos: cada proceso genera su parte del vector en paralelo.

6. Carga la matriz, leyendo el archivo de texto generado por Python. Ojo que cada proceso sólo puede guardar su parte de la matriz, nunca la matriz entera. Guarde cada bloque de la matriz en un arreglo 1D de C++, para lo cual deben usar *dynamic memory allocation* (`https://cplusplus.com/doc/tutorial/dynamic/`).

7. Implemente el método de potencia con MPI bajo el modelo de paso de mensajes.

   a) Se puede usar un número fijo de iteraciones.

   b) Nunca se puede enviar elementos de la matriz entre procesos.

   c) Todas las instrucciones matemáticas (multiplicación matriz por vector, producto putno, norma, etc.) deben ser paralelizadas sobre todos los procesos.

8. Calcule el error del método, es decir, la diferencia entre el valor propio máximo estimado y el valor exacto, lo cual es 10 para esta matriz específica. Explique si tu implementación del método funciona bien.

9. Para analizar la eficiencia paralela, corre el algoritmo con varios números de procesos y mide el tiempo de cómputo.

a) Deben correr el código en el clúster de Ingeniería UC, en la cola de trabajo `full`. ¡No pueden usar el nodo de cabeza para hacer cálculos!

b) Imprime el nombre del procesador en cada proceso de MPI.

c) Elige una dimensión de la matriz y un número de iteraciones adecuada, tal que el código demora un par de minutos.

d) Pueden guardar el tiempo de cálculo en un archivo y analizar la eficiencia paralela en tu computador local, con Excel o Python.

e) Calcule la aceleración (*speedup*) y eficiencia paralela y analice ambos *strong scaling* y *weak scaling*. Explique cómo analizaron el escalamiento fuerte y débil.

f) ¿Qué pasa si se pide tantos procesos que SLURM debe asignar dos nodos distintos?

Evaluación

En esta tarea, deben responder las preguntas en un informe separado, de Word o Latex. Entregue

1. todo el código de C++,
2. el script de SLURM,
3. el archivo con el output del código (con, al menos, los nombres de procesadores y tiempo de cómputo) para el cálculo más grande que hicieron
4. y el informe

en una mapa comprimida a través de Canvas.

Los reglamientos del curso se puede encontrar en Canvas. Se destaca que las tareas deben ser hechas de forma individual. No se puede compartir código entre compañeros, tampoco usar código de fuentes externos salvo la página del curso en Canvas.

Sugerencias

Deben correr el código en el clúster de Ingeniería UC. Algunas sugerencias:

- Ver `https://cluster.ing.puc.cl` para más información sobre el clúster de Ingeniería UC.
- Siempre deben correr el código en los nodos de trabajo, utilizando la cola de trabajo SLURM.
- Si corren el código de Python en el clúster, también debe ser ejecutado detrás de SLURM.

- Pueden revisar la cola de trabajo con `squeue`.
- Si el código quedó pegado, por ejemplo en un deadlock, pueden terminar la tarea con `scancel 12345` con el número cambiado por el identificador del job.
- Para utilizar MPI, deben correr el comando

  ```
  module load mpi/openmpi-x86_64
  ```

  en el terminal de BASH antes de enviar un trabajo a SLURM. Si no, reciben un error `mpic++: command not found`.
- Ojo que hay que pedir la cantidad de procesadores con SLURM. Para multithreading con Python, deben elegir `ntasks=1` y `cpus-per-task=8` para pedir 8 hilos. Para multiprocessing con MPI, deben elegir `ntasks=8` y `cpus-per-task=1` para pedir 8 procesos. Ver Canvas para ejemplos de scripts de SLURM.
- Ojo que en el clúster no pueden elegir el nodo de cómputo: SLURM lo hace para ti. Los nodos tienen hardware distinto y uno puede ser más rápido que el otro.
- En BASH, y por lo tanto también en el script de SLURM, el comando `date` imprime la fecha y hora y el comando `time mpirun -np 2 a.out` imprime el tiempo de cálculo de MPI.

Algunos comandos útiles de BASH para usar en el terminal/console:

- Ordenar los archivos por fecha: `ls -ltr`
- Ver el tamaño de archivos: `du -shc *`
- Mostrar las últimas líneas de un archivo: `tail -f salida.txt` (salir con CTRL+c)