IMT2112 Semestre 2024-2
Tarea 2

Elwin van 't Wout

August 29, 2024

Introducción

El algoritmo PageRank era diseñado para calcular la importancia de páginas de web en el internet y es comunmente utilizado por buscadores como Google. El web es representado como 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. Corren el código de Python disponible en Canvas. Este código guarda una matriz diagonalizable con valores propios reales entre 1 y 10. Los primeros dos números en el archivo son las dimensiones de la matriz. Después, cada

fila en el archivo corresponde a un único elemento de la matriz, contado fila por fila. Si quieren, pueden cambiar el tamaño de la matriz con la variable `ndim`.

2. Ahora deben implementar el método de potencia en paralela con MPI, donde cada proceso tiene un bloque de filas de la matriz. En otras palabras, utilizamos un particionamiento por filas. Todas las instrucciones siguientes deben ser implementadas en C++.

3. Leen el tamaño de la matriz desde el archivo y calculen 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.

4. Inicialicen el vector $\mathbf{b}_0$ con valores igual a uno. Este vector también debe ser particionado sobre los procesos: cada proceso genera su parte del vector en paralelo.

5. Cada proceso necesita su parte de la matriz. Entonces cada proceso debe leer solo una parte del archivo con los elementos de la matriz. Ojo que no se puede guardar toda la matriz en ningún proceso: cada proceso guarda únicamente su parte de la matriz. Guarden los bloques de la matriz en un arreglo 1D de C++, para lo cual deben usar *dynamic memory allocation*.

   - Pueden encontrar un ejemplo de leer archivos con C++ en Canvas.
   - Más información de crear arreglos dinámicos: `https://cplusplus.com/doc/tutorial/dynamic/`.

6. Implementen el método de potencia con MPI bajo el modelo de paso de mensajes.

   - Se puede usar un número fijo de iteraciones.
   - Nunca se puede enviar elementos de la matriz entre procesos.
   - Todas las instrucciones matemáticas (multiplicación matriz por vector, producto interno, norma, etc.) deben ser paralelizados sobre todos los procesos.

7. Calculen 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.

8. Corren el algoritmo con distintos números de procesos y miden el tiempo de cómputo. Discuten la diferencia en tiempo de cálculo cambiando el número de procesos.

   - 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!
   - Imprimen el nombre del procesador en cada proceso de MPI.

- Eligen una dimensión de la matriz y un número de iteraciones adecuada, tal que el código demora un par de minutos.
- Calculen la aceleración (*speedup*) y eficiencia paralela y analicen ambos *strong scaling* y *weak scaling*. Pueden exportar los tiempos de cómputo a Excel o Python para hacer este análisis.
- ¿Qué pasa si se pide tantos procesos que SLURM debe asignar dos nodos distintos?

Evaluación

Entreguen todo el código de C++, el script de SLURM, 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 y deben cumplir las políticas de uso del clúster de Ingeniería UC.

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.
- 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` antes de enviar un trabajo a SLURM.
- 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 son tienen hardware distinto y uno es 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)