Lab3: the power method on GPU

October 9, 2023

1 Introduction

The power method is used to compute (for some matrices) the largest eigenvalue (in absolute value) of a matrix as well as an eigenvector associated to this eigenvalue. This method is described for example here:
`https://en.wikipedia.org/wiki/Power_iteration`
This algorithm, although quite simple, is the key algorithm of real-world applications such as Google's PageRank algorithm or Twitter's recommendation system. The issue is that for large matrices the computation time becomes (too) long. We will therefore deploy this algorithm on GPU to accelerate its computation. Given the GPUs available at Grid'5000, the computations will be done in single precision floating-point arithmetic, and the error threshold will be set to 10^{-5}.

2 Work description

1. We will use the Nvidia P100 GPUs available at Grid'5000. To reserve and access a compute node with such a GPU, you have to run the command:

   ```
   $␣oarsub␣-l␣/cpu=1/gpu=1/core=2/,walltime=2:0:0␣-I␣-p␣"gpu_model␣=␣'Tesla␣P100-PCIE-16GB'"
   ```

   You should end up on a computer named "chifflot-X".

   Run the program:

   ```
   $␣/usr/local/cuda/extras/demo_suite/deviceQuery
   ```

   or the command:

   ```
   $␣nvidia-smi␣-a
   ```

   to display the available information on the GPU present in this computer.

2. Get the following source codes.

   - `sequential.c`: a sequential implementation of the power method. This code generates a specific random matrix of size $n \times n$ (n being given on the command line), then computes an eigenvector (of size n), and finally saves this vector in a file.
   - `checker.c` : source code of a program used to check the eigenvector result.

   One can use $n = 2048$ for short (development) tests, or even $n = 32$ for debugging, and $n = 32768$ for longer (performance) tests.

   All the code (host code and GPU code) can be written in a single file with the `.cu` suffix and compiled with the CUDA `nvcc` compiler with the following options (for P100 GPUs with *compute capability* 6.0):
   `--generate-code arch=compute_60,code=sm_60 -O3`.

   Note that the reference CPU code should then be compiled (using `gcc`) with `-O3` too.

3. How many different CUDA kernels are required to deploy the power method on GPU?

4. Is it necessary to perform CPU-GPU data transfers between these kernels?
   And between multiple iterations of the power method?

5. Write a first (simple) deployment on GPU, using by default 256 threads per block, and compare the GPU performance with the CPU one.

6. How to optimize the GPU performance?

7. (optional) An implementation of the BLAS routines is available in CUDA: CUBLAS (see `https://developer.nvidia.com/cublas`). Write a new version of your code with these CUBLAS routines and compare the performance of your best version without BLAS with this CUBLAS version.