Sabancı University

Faculty of Engineering and Natural Sciences

CS406/CS531
Parallel Computing
Parallel Processing and Algorithms

Spring 2020-2021

Homework #2

Due: 02/05/2021 - 23:59

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You HAVE TO write down the code on your own.
You CANNOT HELP any friend while coding.
Plagiarism will not be tolerated!!

1 Introduction

In this assignment, you will implement a **parallel Sinkhorn-Knopp scaling algorithm on GPU** that is used to convert a given matrix to the *doubly stochastic* form. In this form, each row and column sum equals to one. Definition of the problem, algorithm to implement, and program flow are given in the following sections.

2 Problem

An $n \times n$ matrix $\mathbf{A}$ is said to be scaled with two (unique) positive diagonal matrices $\mathbf{D}_R$ and $\mathbf{D}_C$ such that $\mathbf{D}_R\mathbf{A}\mathbf{D}_C$ is doubly stochastic (that is, the sum of entries in any row and in any column of $\mathbf{D}_R\mathbf{A}\mathbf{D}_C$ is equal to one). The existence of $\mathbf{D}_R$ and $\mathbf{D}_C$ that scale $\mathbf{A}$ to doubly-stochastic form depends on the *decomposability* and *support* of $\mathbf{A}$. In this HW, you are not required to check these properties of $\mathbf{A}$. Your code will be tested only with 0/1 matrices scalable to the doubly-stochastic form.

The Sinkhorn-Knopp algorithm is a well-known method for scaling matrices to doubly stochastic form. This algorithm generates a sequence of matrices (whose limit is doubly stochastic) by normalizing the columns and the rows of the sequence of matrices in an alternating manner. That is during the course of the algorithm, the initial matrix is first normalized such that each column has sum one. Then, the resulting matrix is normalized so that each row has sum one and so on so forth. The details of the algorithm are given in Algorithm 1.

2.1 Compressed Sparse Row Representation

In this HW, you will work with 0/1 sparse matrices. That is most of the matrix entries are 0 and the remaining entries are equal to 1. In practice, the Compressed Sparse Row (CSR) and Compressed Sparse Column representations are used to reduce the space requirement to store the matrices in memory and to enable fast access to the non-zero elements (i.e., nonzeros). For a 0/1 matrix, the CSR (CCS) representation uses two arrays, `adj` and `xadj` (`tadj` and `txadj`) for storing the column (row) indices of the nonzeros within each row (column) of the matrix adjacently. In CSR, the length of the array `adj` is equal to number of nonzeros, and it keeps the column indices for the rows. The array `xadj` keeps the starting index for each row in `adj`. Hence, for a row i, the sub-array of column indices of nonzeros at row i starts with the entry `adj[xadj[i]]` and ends with the entry `adj[xadj[i+1]]` -1. To simplify the implementations and make the previous statement correct for the last row, the length of `xadj` is set to $n+1$, and the last element, `xadj`[n], is set to number of nonzeros. An example CSR and CCS representation for a toy sparse matrix are given in Figure 1.

Similarly, in CCS, `tadj` is an array of size number of nonzeros, and it is used to keep the column indices for the rows. The array `txadj` keeps the starting index for each column in `tadj`. Hence, for the ith column, the first and the last nonzero row indices in `tadj` are `tadj[txadj[i]]` and `tadj[txadj[i+1]]` -1. For the Sinkhorn-Knopp algorithm, you will need both CSR and CCS.

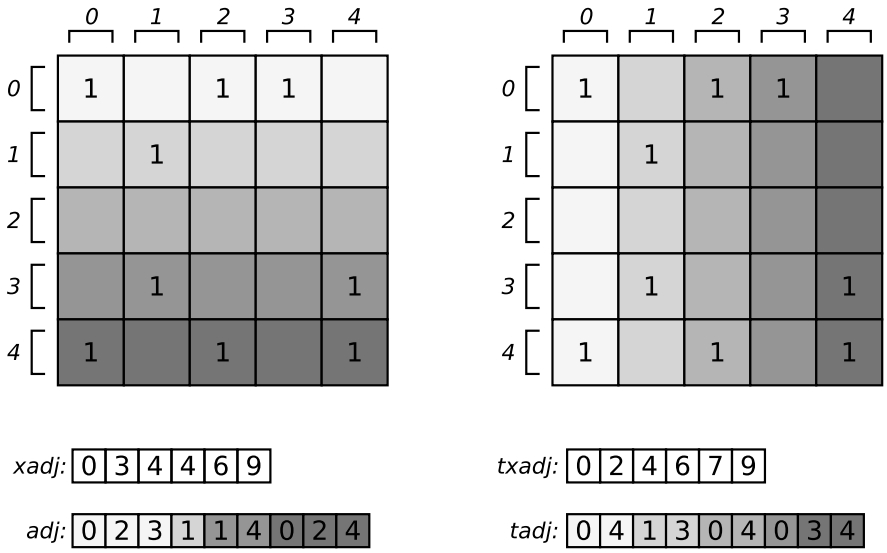

Figure 1: CSR (left) and CCS (right) representations for a sparse matrix.

Algorithm 1: SINKHORN-KNOPP SCALING

Input: $\mathbf{A}$: an $n \times n$ matrix, *iterations*: number of SK iterations
Output: $\mathbf{d}_r, \mathbf{d}_c$: row / column scaling arrays (entries of the diagonal scaling matrices)

for $i \leftarrow 1$ **to** n **do**
  $\mathbf{d}_r[i] \leftarrow 1$
  $\mathbf{d}_c[i] \leftarrow 1$
for $x \leftarrow 1$ **to** *iterations* **do**
  for $i \leftarrow 1$ **to** n **do**
    $rsum \leftarrow \sum_{j \in \mathbf{A}_{i*}} a_{ij} \times \mathbf{d}_c[j]$
    $\mathbf{d}_r[i] \leftarrow 1/rsum$
  for $j \leftarrow 1$ **to** n **do**
    $csum \leftarrow \sum_{i \in \mathbf{A}_{*j}} a_{ij} \times \mathbf{d}_r[i]$
    $\mathbf{d}_c[j] \leftarrow 1/csum$
  $error \leftarrow \max \left\{ abs \left(1 - \left(\sum_{j \in \mathbf{A}_{i*}} \mathbf{d}_r[i] \times a_{ij} \times \mathbf{d}_c[j] \right) \right) : 1 \leq i \leq n \right\}$

3 Program Flow

For this homework, two files are provided to you; `main.cpp` and `kernel.cu`. In the file `main.cpp`, matrices are read and cuda code is called. You should not change this file and write your driver and kernel code in `kernel.cu`. You also need to compile these files together. Your program should accept two arguments: `filepath` and `number of iterations`. Moreover, you need to report error at each iteration and time for each run. A sample run should be as following:

```
$>> ./scalesk /gandalf/data/CS406_531_HW2/nlpkkt240.mtxbin 5

./try /gandalf/data/CS406_531_HW2/nlpkkt240.mtxbin 8
fname:  /gandalf/data/CS406_531_HW2/nlpkkt240.mtxbin
iter 0 - error 0.702533
iter 1 - error 0.3595
iter 2 - error 0.241694
iter 3 - error 0.182051
iter 4 - error 0.146013
GPU scale -- Time:  1.624871 s.
```

You can find the matrices you will use for this homework under `/gandalf/data/CS406_531_HW2`. Error values for sample runs are provided under `/gandalf/data/CS406_531_HW2/sample_runs`. You can check the correctness of your implementation from these files. There could be small differences ($< 1\%$) between sample runs and your runs. Moreover, performance of a sample code is also provided to benchmark your implementations:

Matrix	1t	8t	GPU
vas_stokes_1M	2.79	0.44	0.60
vas_stokes_4M	10.98	1.80	2.12
stokes	28.69	6.69	6.18
nlpkkt240	30.78	8.19	6.46

Table 1:

Note that this is a benchmark implementation to give you an idea and your run time may be faster or slower than this implementation. But, **try to optimise your code as much as you can because part of your submission will be evaluated based on its relative performance compared to the rest of the class.** Moreover, run times in your report should be consistent to the run times obtained while grading. Therefore, take your runs on the gandalf computer and don't forget to include run times and speed-up values for your implementation in your report. Measure the GFLOPs of your algorithm and report it. Try to use different parallelization techniques for GPU (e.g., thread-based parallelisation, warp-based parallelisation, shared memory, coalesced access etc.), report the timings and explain the differences in detail.

Note: You must change the version of `gcc` in gandalf by running `module load gcc/7.5.0`. And load CUDA by running `module load cuda/10.0`

4 What and Where to Submit (PLEASE READ, IMPORTANT)

Please don't forget to submit your code and the report together. Your **REPORT** must be a **pdf** file (preferable prepared by LaTeX but MS Word converted pdf's are also OK). It must contain the description of the optimizations you implemented, i.e., it must explain how you improved the performance, what was the timings before and after. You must do this for both of the –O0 and –O3 optimization options. Please see above what else do you need to include in the report.

The grading process is not automatic. However, the students are expected to strictly follow the guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade. The name of your source code file that contains your program must be **scalesk_hw1.cpp**. Similarly the report must be named as **report_hw1.pdf**.

Put both of these files into a folder named

SUCourseUserName_YourLastname_YourName_HWnumber

Your SUCourse user name is actually your SUNet username that is used for checking sabanciuniv e- mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the folder name must be:

cago_Caglayan_Ozbugsizkodyazaroglu_hw1

Do not add any other character or phrase to the folder name. Make sure that it contains the last version of the source code and the report. Compress this folder using a zip program. Please use "zip" compression. **"rar" or another compression mechanism is NOT allowed. Please make sure that you include both of the files in the compressed folder.**

You will receive no credits if your compressed folder does not expand or it does not contain the correct files. The name of the zip file should be as follows:

SUCourseUserName_YourLastname_YourName_HWnumber.zip

For example zubzipler_Zipleroglu_Zubeyir_hw1.zip is a valid name, but

hw1_hoz_HasanOz.zip, HasanOzHoz.zip

are **NOT** valid names. **Submit via SUCourse ONLY!** You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!

CS406-531 Team (Fatih Taşyaran, Kamer Kaya)