COL380: Assignment 1

Sparse Matrix Operations Using Shared Memory

Due Feb 13^{th} 2023

1 Introduction

Matrix multiplication is a common problem in Linear Algebra. Matrix multiplication between two matrices A_{m*n} and B_{n*k} is $C_{m*k} = A_{m*n} * B_{n*k}$. It finds application in many fields, including scientific simulations, machine learning, and social media analysis, and can take various forms.

Let a_{ij} be the j^{th} element of i^{th} row of matrix A, b_{ij} be the j^{th} element of i^{th} row of matrix B, and c_{ij} be the j^{th} element of i^{th} row of resultant matrix C where $C = A \cdot B$.

$$c_{ij} = \sum_{p=0}^{n-1} a_{i,p} * b_{p,j} \qquad \forall \qquad 0 \le i < m \;\; \& \;\; 0 \le j < k$$

Some applications require a more general form of the definition above, where the inner operation "*" and the outer operation "+" can be general binary operations, usually associative and often commutative. For example, the inner operation could be "+" and outer can be "min", making matrix $C = A \odot B$

$$c_{ij} = \min_{p=0}^{n-1}(a_{i,p} + b_{p,j}) \qquad \forall \qquad 0 \le i < m \;\; \& \;\; 0 \le j < k$$

In particular, an $n * n$ adjacency matrix A of a graph with n vertices has a_{ij} = the weight of the edge between vertex i and vertex j for $i < j$. Note that a_{ji} is not stored if $i < j$ but equals a_{ij}. Multiplications of such graphs can provide many insights, e.g., in spectral analysis. Often, such graphs of interest are highly sparse: each vertex has only a small number of neighbors.

Let B be an $n * n$ square matrix such that

$$B = A^P : A \odot A \odot A.. \text{ P times}$$

.

2 Problem Statement

Your goal is to compute the matrix $B = A \odot A$, given a **symmetric** sparse matrix A, and generic inner and outer operations without incurring any arithmetic overflow. In other words, the final elements of matrix B have to be

$$b'_{ij} = min(b_{ij}, \;\; MAX\_VAL)$$

where MAX_VAL is $2^{16} - 1$, and b_{ij} is the computed value. Your code will be based on OpenMP tasks, and should scale well. (Tests will go up to 48 cores.)

3 Input Format

Assume that A_{n*n} is subdivided into blocks of $m*m$ elements. Only the blocks with at least one non-zero element are given in the input file. The other blocks are assumed to have all 0s. For example, we can divide a $100*100$ square matrix into $5*5$ element blocks, i.e. 400 total blocks, which themselves have a $20*20$ configuration, allowing us to index blocks as (i,j). Further the input is provided in upper triangular format. i.e. $block_{ij}$ is provided in input iff $i \leq j$

The input file is a little-endian binary file containing the input in the following format:

1. First 4 bytes represent n (dimension of the matrix)
2. next 4 bytes represent m (dimension of the block)
3. next 4 bytes represent numbers of blocks given as input(say k)
4. now, each block is represented as
   (a) First 4 bytes represent the i^{th} index of the block
   (b) next 4 bytes represent the j^{th} index of the block
   (c) next m^2 bytes represent the block in a row-major order, where 1 byte corresponds to the 1 element(cell) of the block. (Values are < 256.)

Note:-

1. The 4 bytes representation of a number has the most significant byte at the end.
2. Inner and Outer functions must be generic in the matrix operation. library.hpp and library.so files are given with sample inner and outer functions. You must use this library while solving the problem. While testing, library.hpp file will remain same but library.so file can be replaced with some other library, meaning a different operation and different cost. Your code should remain efficient no matter how long these operations take.

4 Output Format

Output the resultant matrix in a binary file using the following format. Only the blocks with at least one non-zero element and lying in the upper triangular region must be output.

1. First 4 bytes represent n (dimension of the matrix). This should be the same as the input file.
2. next 4 bytes represent m (dimension of the block). This should be the same as the input file.
3. next 4 bytes represent numbers of blocks(say k).
4. now, each block is represented as
   (a) First 4 bytes represent the i^{th} index of the block
   (b) next 4 bytes represent the j^{th} index of the block

(c) next $2 * m^2$ bytes represent the block where 2 bytes corresponds to the 1 element(cell) of the block. (Recall, the stored value $b' < 2^{16}$.)

Your report should list your journey to finding the best solution in the text file named Readme, and a pdf file showing your final task graph. The Readme file should be in text and must follow the following format:
< Begin format > ——————————
Number of approaches:
Approach 1
Approach 2
...
...
Final Approach
Final scalability analysis
< End format > ——————————
Format for each approach: (the first digit is the approach number)
1:Idea: your text
1:Why attempted: your text
1:Results (Specify speedup): your text
1:Drawback: your text
——————————
Format for Final scalability analysis: perfomance table followed by a paragraph of your commentary on it. Performance table is a speed vs core table with increasing matrix sizes in csv format: (the following is formatted in latex for clarity)

Non-0 input blocks	Non-0 output blocks	2 cores	4 cores	8 cores	16 cores
2^{10}					
2^{15}					
2^{20}					
2^{25}					

Table 1: Time in ms, two decimal points

5 COL380 vs COV880

- For COL380, only $P = 2$ needs to be implemented.
- For COV880, IO latency should be hidden.

6 Submission Instructions

Submit a zip file (<EntryNo>_A1.zip) that contains the following directory structure:

```
2020CS10XXX_A1/
|-- Makefile
|-- readme.txt
|-- task.pdf
|-- other code files
```

The following commands need to be executed to run your program:

```
make
```

This should create an executable(exec) for your program. After this, run the executable with proper command line arguments using the following command:

```
./exec inputFile.bin outFile.bin
```

This should dump your output into the outFile.bin file. A sample example of input and output is given below:

Input files :

1. Library Files:
   (a) `https://www.cse.iitd.ac.in/~cs5180411/col380/library.hpp`
   (b) inner-outer pair 1 : `https://www.cse.iitd.ac.in/~cs5180411/col380/library.so`
   (c) inner-oouter pair 2 : `https://www.cse.iitd.ac.in/~cs5180411/col380/library2.so`
2. Sample Input file : `https://www.cse.iitd.ac.in/~cs5180411/col380/input`
   This is a comparatively small input file. Larger samples will also be provided. You may create your own as well.