Parallel & Distributed Computer Systems 2023 Homework Report

Evangelos Kyriakos

November 23, 2023

Abstract

This is an implemantation of a sparse matrix multiplication problem using parallisation techniques. Here, we parallelize the matrix multiplication using Pthreads (POSIX), OpenMP and OpenCilk writen in C.

1 Background

A major part of this project is based on deciding and concluding on the proper format to store the matrices in. Sparse matrices are created to avoid large memory overhead. This naturally bounds our potential storage forms. Commonly used formats include Compressed Sparse Row (CSR), Compressed Sparse Column (CSC), Rutherford-Boeing, and Market Matrix. Of these CSR and CSC lend themselves most naturally to the row/column based accesses that characterize matrix multiplication. But specifically since we are focusing on row indices multiplication, the final sparse matrix format is that of a CSR.

On its own, dense matrix multiplication is naturally memory-bound. By representing the matrix as a CSR, we add additional memory accesses by the virtue that we cannot simply index into an array, but instead need to search. Many of the standard matrix multiplication techniques can potentially apply but have this additional overhead.

It is important to speedup sparse matrix multiplication as there are a variety of applications that often rely on successive multiplications. Faster compute is eccencial for many networking, natural language processing, and GPU acceleration are few of the numberous implementations.

2 Problem Approach

In this implementation, the program takes as input a single adjecency matrix, commonly provided on a COO format from the Matrix Market provider. If no adjecency matrix is given or isn't loaded properly, a randomly generated adjacency martix may be produced, where with a given row and column lenght, edge probability density, and a random generator seed, arbirtary matrices are produced.

Given the adjecency matrix a randomly generated configuration is produced per each node, mapping all nodes to a cluster state space.

Then $\Omega_{n\times c}$ configuration matrices such that $\Omega_{ij} = 1$ if node i belongs to cluster j is created per the CSR adjecency matrix A. Thus the final minor graph compute of $M = \Omega^T A\Omega$ is given.

The adjacency matrix generator, has the further options of dynamic of the matrix size, edge probability of the said matrix, and a RNG factor.

It is worth mentioning that the choise between a CSR / CSC matrix format is symmetric. For the purposes of the matrix multiplication multi-threaded applications, the CSR format was chosen for the expected row product element oparations. Matrix Market .mtx inputs is that of one argument of the file root directory location. There are some hand picked (.mtx) sparse matrices at the GitHub repo. The application build requires the Matrix Market (mmio.h) file. Just like every other sparse Matrix Market files, the non zero coordinates are given in a COO format, which requires the appropriate convertion to a CSR for our cases, as well as loading the value of one, to display the adjacancy matrix architecture.

3 Coding Implementation

The software was written in C, to make use of C's low level control over the CSR matrix attributes. In C, n efficient memory utilization of the CSR format stores sparse matrices

in a memory-efficient way. Performance, memory management, high portability across architectures, and high parallelization control, led to this implementation.

The main program executable has been split to the seperate programs library case usages. The PThreads and OpenMP can be compiles with either the gcc or clang using the appropriate flags, while OpenCilk, requires the a proper source build of the OpenCilk flag at clang.

3.1 Control Flow

In evey one of the programs multithreading mode, the control flow stays the simmilar, with the following variable inputs

- Adjacency matrix generator / mmio read of .mtx input file
- Random cluster generation of all nodes
- Configuration matrix generation by given adjacency matrix and node cluste array inputs
- Configuration matrix transpose calculation
- Multi-threades multiplication strategy
- Correspoding multiplication strategy caclulation with set timer

3.2 Notes

The code although can handle expected inputs, like adjacency matrix generation or number of threads choise, many non-releasting or huge inputs aren't garrented to work. Furthermore, many compiler flag options aren't guarrented to always work, while the code suffers with various memory leeks. Please respond to any terminal compilation or runtime errors.

4 Results

Overall there was a success in speeding up the matrix multiplication as seen by the speed up in the graph below. I chose speedup timing using a sys/time machine timer, which takes the runtimes for all threads spawned. Likewise I tested the following results on the auto-generator sparse matrix results. All simullated results are based on the set generator, with the following traits; Of course the sparse matrix size, is found on the x-axis which represents the number of rows in the matrix (all generated matrices were square matrices). Simillary for any of the size an edge density probability of 0.05 was use of the matrices above size 50. All non zero elements have a value of 1.

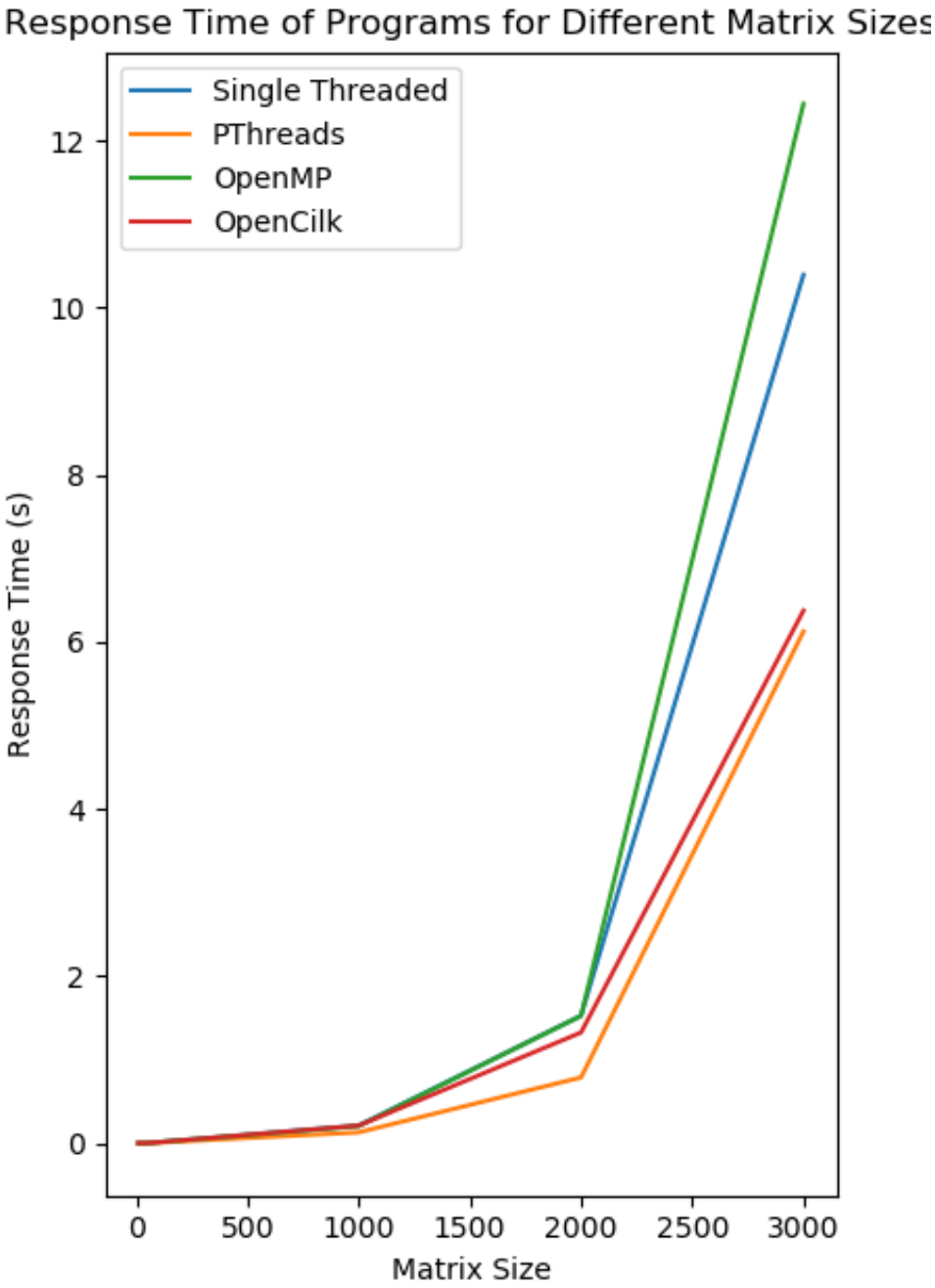

Figure 1: CSR Parallel multiplication response times per matrix size