**CMPE 478**

**Homework 2, Fall 2022**

**Kadir Gökhan Sezer**

**Demet Yayla**

The algorithm we use is a simple version of the algorithm described in detail at this url(http://infolab.stanford.edu/~backrub/google.html). This application, which seems complicated, is not very difficult for people with graph theory knowledge. First of all, if we try to understand the purpose of the algorithm, 1990 and 2000 were the years when the web was born. With the unexpectadly fast growth on its content volume, it was understood that there was a need for search engines. The lack of technological growth (with respect to hardware) required some advance algorithms to emerge on the topic: In 1994, a few professors at Stanford University came up with a ranking algorithm, the very algorithm Google utilized in its foundation phase.

Diagram

Description automatically generated

<https://delante>.co/definitions/pagerank/

The purpose of this algorithm is to determine the order of site suggestions that an ordinary user will come across while browsing the internet with the aim of benefiting the user with the most related content it has to the spesific query. The app simply gives each site a value it has calculated. It then sorts the sites according to this value. This value is directly proportional to the links it receives. In other words, the more links a site gets, the more valuable it becomes according to this algorithm.

In essence, we used a simpler variation of the algorithm. The original formulae is as below:

Text

Description automatically generated

<https://delante.co/definitions/pagerank/>

The simpler version we implemented is as below:

Text, letter

Description automatically generated

<https://www.cmpe.boun.edu.tr/tr/people/can.ozturan>

In the version we use, the algorithm has been simplified and made short-term calculations and analyzes possible.

The pseudocode for the PageRank algorithm.

Text

Description automatically generated

<https://www.ccs.neu.edu/home/daikeshi/notes/PageRank.pdf>

**CSR**

Chart, scatter chart

Description automatically generatedIn this Project, we need to use CSR matrix to keep the data in RAM. If we not, there would be no enough space to keep the data in RAM. What is CSR matrix simply is to keep only nonzero values.

In numerical analysis and scientific computing, a sparse matrix or sparse array is a matrix in which most of the elements are zero.[1] There is no strict definition regarding the proportion of zero-value elements for a matrix to qualify as sparse but a common criterion is that the number of non-zero elements is roughly equal to the number of rows or columns. By contrast, if most of the elements are non-zero, the matrix is considered dense.[1] The number of zero-valued elements divided by the total number of elements (e.g., m × n for an m × n matrix) is sometimes referred to as the sparsity of the matrix.

Conceptually, sparsity corresponds to systems with few pairwise interactions. For example, consider a line of balls connected by springs from one to the next: this is a sparse system as only adjacent balls are coupled. By contrast, if the same line of balls were to have springs connecting each ball to all other balls, the system would correspond to a dense matrix. The concept of sparsity is useful in combinatorics and application areas such as network theory and numerical analysis, which typically have a low density of significant data or connections. Large sparse matrices often appear in scientific or engineering applications when solving partial differential equations.

When storing and manipulating sparse matrices on a computer, it is beneficial and often necessary to use specialized algorithms and data structures that take advantage of the sparse structure of the matrix. Specialized computers have been made for sparse matrices,[2] as they are common in the machine learning field.[3] Operations using standard dense-matrix structures and algorithms are slow and inefficient when applied to large sparse matrices as processing and memory are wasted on the zeros. Sparse data is by nature more easily compressed and thus requires significantly less storage. Some very large sparse matrices are infeasible to manipulate using standard dense-matrix algorithms.

Source: <https://en.wikipedia.org/wiki/Sparse_matrix#Compressed_sparse_row_(CSR,%20_CRS_or_Yale_format)>

The implementation we used:

Text

Description automatically generated

The sequential code:

#include <iostream>

#include <fstream>

#include <set>

#include <iterator>

#include <map>

#include <queue>

#include <vector>

#include <string>

#include <sstream>

#include <unordered\_map>

#include <utility>

#include <cstdio>

#include <map>

#include <algorithm>

#include <cmath>

#include <omp.h>

#include <chrono>

#include <mpi.h>

#include <metis.h>

using namespace std;

/\*

rth row and jth column (r and j starting from 0)

a. increase row\_begin(r+1) by 1

b. sum from 0 to r-1. values[sum+1] is the new node's position

c. col\_indices[sum+1] is its equivalent column reference

3 r 4 c

in-degree --> row sum

oudegree --> column sum

If there is an edge from k to l, the kth column lth row is 1

\*/

unordered\_map<int, vector<int>> all\_edges\_new;

#define FILE "graph.txt"

#define DEBUG\_FILE\_READ 0

#define CSR\_PRINT 0

class CSR{

public:

vector<int> row\_begin;

CSR(int nodeNum);

double\* values;

vector<int> col\_indices;

};

CSR::CSR(int nodeNum){

this->row\_begin = vector<int>(nodeNum+1);

this->col\_indices = vector<int>();

}

CSR\* csr;

int main(int arg, char\* argv[])

{

// For MPI

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

// Inıtialize the variables

string first,second;

int index = 0;

// This holdes the all edges as source and destination

vector<pair<int, int> > all\_edges;

//This holds the all nodes as name and index

unordered\_map<string, int> dictionary;

unordered\_map<int, vector<int> > all\_edges\_new;

unordered\_map<int, int> out\_arrows;

freopen(FILE, "r", stdin); // read from file

//READING FILE

while (cin >> first >> second)

{ // We first used vector<vector<int>> all\_edges, but it was too slow.

// So we used vector<pair<string, string>> all\_edges, it is faster to read.

// Since it is hard to compare string than int, we used dictionary to map string to int.

if (dictionary.find(first) == dictionary.end()) { // Add the first node to dictionary if it is not in the dictionary

dictionary.emplace(first, index);

index++; // Increase the index of the dictionary items

}

if (dictionary.find(second) == dictionary.end()) { // Add the second node to dictionary if it is not in the dictionary

dictionary.emplace(second, index);

index++;

}

if (out\_arrows.find(dictionary[first]) == out\_arrows.end()) { // Add the first node to dictionary if it is not in the dictionary

out\_arrows.emplace(dictionary[first], 1);

}

else {

out\_arrows[dictionary[first]]++;

}

//all\_edges.emplace\_back(dictionary[first], dictionary[second]);

if (find(all\_edges\_new[dictionary[second]].begin(), all\_edges\_new[dictionary[second]].end(), dictionary[first])==all\_edges\_new[dictionary[second]].end()) {

// it is to make the incoming edges unique, i.e it is as if set.

all\_edges\_new[dictionary[second]].push\_back(dictionary[first]);

}

// Since all of them added to dictionary on the first if,

// we do not need to check for first again.

/\*

if (dictionary.find(first) == dictionary.end()) {

dictionary.emplace(first, dictionary.size());

}

\*/

}

fclose(stdin); //Close the file

//READIN FILE ENDS

printf("--> File is read\n");

int n\_number\_of\_nodes = dictionary.size(); // Number of nodes

printf("--> Out\_arrows are calculated\n");

csr = new CSR(n\_number\_of\_nodes);

csr->row\_begin[0] = 0;

int priorLen = 0;

int priorNonzero = 0;

if (rank == 0) {

for(int i = 0; i < n\_number\_of\_nodes; i++){

if(all\_edges\_new.find(i) == all\_edges\_new.end()){

csr->row\_begin[i] = 0;

}else{

vector<int>\* adjList = &all\_edges\_new[i];

csr->row\_begin[i] = priorNonzero + priorLen;

priorNonzero = priorNonzero + priorLen;

priorLen = adjList->size();

sort(adjList->begin(), adjList->end());

csr->col\_indices.insert(csr->col\_indices.end() , adjList->begin(), adjList->end());

}

}

csr->row\_begin[n\_number\_of\_nodes] = priorNonzero + priorLen;

int filledSize = csr->col\_indices.size();

csr->values = new double[filledSize];

for(int i = 0; i < filledSize; i++){

if(out\_arrows[csr->col\_indices[i]]==0){

printf("Error: out\_arrows[csr->col\_indices[i]]==0");

continue;

}

csr->values[i] = 1. / out\_arrows[csr->col\_indices[i]];

}

}

vector<double> r0(n\_number\_of\_nodes, 1.0); // This holds the initial rank of the nodes

vector<double> r\_next(n\_number\_of\_nodes, 0.0); // This holds the next iteration of the pagerank

int iterations = 0; // This holds the number of iterations

double alpha = 0.20; // Alpha value

double diff = 0.0; // Difference between the r0 and r\_next

double epsilon = 0.000001; // Epsilon value

//Calculate the pagerank

double x = (1 - alpha) \* (1. / n\_number\_of\_nodes); //!!

double beginTime = omp\_get\_wtime();

for (int i = 0; i < n\_number\_of\_nodes; i++)

r0[i] = 1.0 / n\_number\_of\_nodes;

while(true)

{

iterations++;

diff = 0.0;

y = 1;

for (int i = 0; i < n\_number\_of\_nodes; i++)

{

r\_next[i] = 0.0;

int y=1;

for (int j = csr->row\_begin[i]; j < csr->row\_begin[i + 1]; j++)

{

r\_next[i] += csr->values[j] \* r0[csr->col\_indices[j]];

}

r\_next[i] = (x + (alpha)\*r\_next[i])/n\_number\_of\_nodes;

diff += abs(r\_next[i] - r0[i]);

}

r0 = r\_next;

printf("Iteration: %d, Diff: %.\*e\n", iterations,10, diff);

if (diff<epsilon)

break;

}

double endTime = omp\_get\_wtime();

printf("--> Pagerank is calculated\n");

printf("Number of iterations: %d\n", iterations);

printf("Top 5 nodes: \n");

vector<pair<double, int>> top5;

for (int i = 0; i < n\_number\_of\_nodes; i++)

top5.emplace\_back(r\_next[i], i);

sort(top5.begin(), top5.end(), greater<pair<double, int>>());

for(auto it = dictionary.begin(); it != dictionary.end(); ++it){

for (int i = 0; i < 5; i++) {

if(it->second == top5[i].second){

printf("%.\*e ", 10,top5[i].first);

cout << it->first<<endl;

}

}

}

return 0;

}

Details of the machine we used:

Graphical user interface

Description automatically generated

Graphical user interface, text

Description automatically generated

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  | |  | | Essentials | | Product Collection,9th Generation Intel® Core™ i7 Processors | | Code Name,Products formerly Coffee Lake | | Vertical Segment,Mobile | | Processor Number,i7-9750H | | Status,Launched | | Launch Date,Q2'19 | | Lithography,14 nm | | Recommended Customer Price,$395.00 | |  | | CPU Specifications | | Total Cores,6 | | Total Threads,12 | | Max Turbo Frequency,4.50 GHz | | Processor Base Frequency,2.60 GHz | | Cache,12 MB Intel® Smart Cache | | Bus Speed,8 GT/s | | TDP,45 W | | Configurable TDP-down,35 W | |  | | Supplemental Information | | Embedded Options Available,No | | Datasheet,View now | |  | | Memory Specifications | | Max Memory Size (dependent on memory type),128 GB | | Memory Types,DDR4-2666 | LPDDR3-2133 | | Max # of Memory Channels,2 | | Max Memory Bandwidth,41.8 GB/s | | ECC Memory Supported ‡,No | |  | | Processor Graphics | | Processor Graphics ‡,Intel® UHD Graphics 630 | | Graphics Base Frequency,350 MHz | | Graphics Max Dynamic Frequency,1.15 GHz | | Graphics Video Max Memory,64 GB | | Graphics Output,eDP/DP/HDMI/DVI | | 4K Support,Yes | at 60Hz | | Max Resolution (HDMI)‡,4096 x 2304@30Hz | | Max Resolution (DP)‡,4096 x 2304@60Hz | | Max Resolution (eDP - Integrated Flat Panel)‡,4096 x 2304@60Hz | | Max Resolution (VGA)‡,N/A | | DirectX\* Support,12 | | OpenGL\* Support,4.5 | | Intel® Quick Sync Video,Yes | | Intel® InTru™ 3D Technology,Yes | | Intel® Clear Video HD Technology,Yes | | Intel® Clear Video Technology,Yes | | # of Displays Supported ‡,3 | | Device ID,0x3E9B | |  | | Expansion Options | | PCI Express Revision,3.0 | | PCI Express Configurations ‡,Up to 1x16 | 2x8 | 1x8+2x4 | | Max # of PCI Express Lanes,16 | |  | | Package Specifications | | Sockets Supported,FCBGA1440 | | Max CPU Configuration,1 | | TJUNCTION,100°C | | Package Size,42mm x 28mm | |  | | Advanced Technologies | | Intel® Optane™ Memory Supported ‡,Yes | | Intel® Speed Shift Technology,Yes | | Intel® Thermal Velocity Boost,No | | Intel® Turbo Boost Technology ‡,2.0 | | Intel® Hyper-Threading Technology ‡,Yes | | Intel® Transactional Synchronization Extensions,No | | Intel® 64 ‡,Yes | | Instruction Set,64-bit | | Instruction Set Extensions,Intel® SSE4.1 | Intel® SSE4.2 | Intel® AVX2 | | Intel® My WiFi Technology,Yes | | Idle States,Yes | | Enhanced Intel SpeedStep® Technology,Yes | | Thermal Monitoring Technologies,Yes | | Intel® Flex Memory Access,Yes | | Intel® Identity Protection Technology ‡,Yes | |  | | Security & Reliability | | Intel® AES New Instructions,Yes | | Secure Key,Yes | | Intel® Software Guard Extensions (Intel® SGX),Yes with Intel® ME | | Intel® Memory Protection Extensions (Intel® MPX),Yes | | Intel® OS Guard,Yes | | Intel® Trusted Execution Technology ‡,No | | Execute Disable Bit ‡,Yes | | Intel® Stable IT Platform Program (SIPP),No | | Intel® Virtualization Technology (VT-x) ‡,Yes | | Intel® Virtualization Technology for Directed I/O (VT-d) ‡,Yes | | Intel® VT-x with Extended Page Tables (EPT) ‡,Yes | |
|  |

<https://www.intel.com.tr/content/www/tr/tr/products/sku/191045/intel-core-i79750h-processor-12m-cache-up-to-4-50-ghz/specifications.html>

This project consists of two sections. We run the algorithm we mentioned above in parallel with two different methods. These methods are:

1. MPI

Diagram

Description automatically generated

### What is the message passing interface (MPI)?

The message passing interface (MPI) is a standardized means of exchanging messages between multiple computers running a parallel program across distributed memory.

In parallel computing, multiple computers – or even multiple processor cores within the same computer – are called nodes. Each node in the parallel arrangement typically works on a portion of the overall computing problem. The challenge then is to synchronize the actions of each parallel node, exchange data between nodes, and provide command and control over the entire parallel cluster. The message passing interface defines a standard suite of functions for these tasks. The term message passing itself typically refers to the sending of a message to an object, parallel process, subroutine, function or thread, which is then used to start another process.

MPI isn't endorsed as an official standard by any standards organization, such as the Institute of Electrical and Electronics Engineers (IEEE) or the International Organization for Standardization (ISO), but it's generally considered to be the industry standard, and it forms the basis for most communication interfaces adopted by parallel computing programmers. Various implementations of MPI have been created by developers as well.

MPI defines useful syntax for routines and libraries in programming languages including Fortran, C, C++ and Java.

### Benefits of the message passing interface

The message passing interface provides the following benefits:

* **Standardization.** MPI has replaced other message passing libraries, becoming a generally accepted industry standard.
* **Developed by a broad committee.** Although MPI may not be an official standard, it's still a general standard created by a committee of vendors, implementors and users.
* **Portability.**MPI has been implemented for many distributed memory architectures, meaning users don't need to modify source code when porting applications over to different platforms that are supported by the MPI standard.
* **Speed.** Implementation is typically optimized for the hardware the MPI runs on. Vendor implementations may also be optimized for native hardware features.
* **Functionality.** MPI is designed for high performance on massively parallel machines and clusters. The basic MPI-1 implementation has more than 100 defined routines.

Some organizations are also able to offload MPI to make their programming models and libraries faster.

These notes and the picture are taken from :

(https://www.techtarget.com/searchenterprisedesktop/definition/message-passing-interface-MPI)  
Thank the techtarget team.

1. Thrust

Logo

Description automatically generated

Thrust is a C++ template library for CUDA based on the Standard Template Library (STL). Thrust allows you to implement high performance parallel applications with minimal programming effort through a high-level interface that is fully interoperable with CUDA C.

Thrust provides a rich collection of data parallel primitives such as scan, sort, and reduce, which can be composed together to implement complex algorithms with concise, readable source code. By describing your computation in terms of these high-level abstractions you provide Thrust with the freedom to select the most efficient implementation automatically. As a result, Thrust can be utilized in rapid prototyping of CUDA applications, where programmer productivity matters most, as well as in production, where robustness and absolute performance are crucial.

This document describes how to develop CUDA applications with Thrust. The tutorial is intended to be accessible, even if you have limited C++ or CUDA experience.

Thrust provides STL-like templated interfaces to several algorithms and data structures designed for high performance heterogeneous parallel computing:

Text

Description automatically generated

Chart, bar chart

Description automatically generated

Thrust is a powerful library of parallel algorithms and data structures. Thrust provides a flexible, high-level interface for GPU programming that greatly enhances developer productivity. Using Thrust, C++ developers can write just a few lines of code to perform GPU-accelerated sort, scan, transform, and reduction operations orders of magnitude faster than the latest multi-core CPUs. For example, the thrust::sort algorithm delivers 5x to 100x faster sorting performance than STL and TBB.

<https://docs.nvidia.com/cuda/thrust/>

<https://developer.nvidia.com/thrust>

The difference between the sequential code and the thrust-parallel version:

In the thrust-paralleized version we need to transfer the datas to the device since :

“A thrust device of the kind in which an axial thrust is produced by relative angular movement between two substantially parallel thrust plates between which balls or rollers co-operate with inclined ramp surfaces, for use in a spreading type disc brake for example. Deformable means are provided between a ramp member and a thrust plate carrying it to allow the ramp member to be displaced axially on initial actuation of the device to even out the load between the balls or rollers for subsequent actuations. The deformable means is provided by having a localized area of contact between the wedge member and its thrust plate.”

Source: <https://patents.google.com/patent/US3964806>

Sequential:

class CSR{

public:

vector<int> row\_begin;

CSR(int nodeNum);

double\* values;

vector<int> col\_indices;

};

vector<double> r0(n\_number\_of\_nodes, 1.0);

vector<double> r\_next(n\_number\_of\_nodes, 0.0);

double diff = 0.0;

Thrust:

thrust::device\_vector<double> values=csr->values;

thrust::device\_vector<double>r0(n\_number\_of\_nodes,1/n\_number\_of\_nodes);

thrust::device\_vector<double> r\_next(n\_number\_of\_nodes, 0.0);

thrust::device\_vector<double> diff(n\_number\_of\_nodes, 0.0);

Until now, we transfered the data to the device. Now it is time to calculate the differences.

Sequential:

while(true)

{

iterations++;

diff = 0.0;

y = 1;

for (int i = 0; i < n\_number\_of\_nodes; i++)

{

r\_next[i] = 0.0;

int y=1;

for (int j = csr->row\_begin[i]; j < csr->row\_begin[i + 1]; j++)

{

r\_next[i] += csr->values[j] \* r0[csr->col\_indices[j]];

}

r\_next[i] = (x + (alpha)\*r\_next[i])/n\_number\_of\_nodes;

diff += abs(r\_next[i] - r0[i]);

}

r0 = r\_next;

printf("Iteration: %d, Diff: %.\*e\n", iterations,10, diff);

if (diff<epsilon)

break;

}

Thrust:

while(true)

{

iterations++;

for (int i = 0; i < n\_number\_of\_nodes; i++)

{

auto p\_iter = thrust::make\_permutation\_iterator(r\_next.begin(),

col\_indices.begin()+row\_begin[i]);

auto i\_prod = thrust::inner\_product(row\_begin[i] + values.begin(), values.begin()

+ rowBegin[i + 1], p\_iter, 0.0);

r\_next[i] = ((i\_prod\*alpha)+x)/n\_number\_of\_nodes;

}

double result = sthrust::transform\_reduce(r\_next.begin(), r\_next.end(), r0.begin(),

diff.begin(), abs\_<double>(), 0.0, thrust::plus<double>());

if (result <= epsilon)

break;

else {

thrust::copy(r\_next.begin(), r\_next.end(), r0.begin());

}

printf("Iteration: %d, Diff: %.\*e\n", iterations,10, diff);

}

For the calculations, we need to use the API’s of the NVIDIA Cuda. We can see from the codes that calculations are done in devices.

* **make\_permutation\_iterator:**

make\_permutation\_iterator creates a permutation\_iterator from an ElementIterator pointing to a range of elements to "permute" and an IndexIterator pointing to a range of indices defining an indexing scheme on the values.

Parameters

An ElementIterator pointing to a range of values.

An IndexIterator pointing to an indexing scheme to use on e.

Returns

A new permutation\_iterator which permutes the range e by i.

* **inner\_product:**

inner\_product calculates an inner product of the ranges [first1, last1) and [first2, first2 + (last1 - first1)).

Specifically, this version of inner\_product computes the sum init + (\*first1 \* \*first2) + (\*(first1+1) \* \*(first2+1)) + ...

The algorithm's execution is parallelized as determined by exec.

Parameters

exec The execution policy to use for parallelization.

first1 The beginning of the first sequence.

last1 The end of the first sequence.

first2 The beginning of the second sequence.

init Initial value of the result.

Returns

The inner product of sequences [first1, last1) and [first2, last2) plus init.

* **transform\_reduce:**

transform\_reduce fuses the transform and reduce operations. transform\_reduce is equivalent to performing a transformation defined by unary\_op into a temporary sequence and then performing reduce on the transformed sequence. In most cases, fusing these two operations together is more efficient, since fewer memory reads and writes are required.

transform\_reduce performs a reduction on the transformation of the sequence [first, last) according to unary\_op. Specifically, unary\_op is applied to each element of the sequence and then the result is reduced to a single value with binary\_op using the initial value init. Note that the transformation unary\_op is not applied to the initial value init. The order of reduction is not specified, so binary\_op must be both commutative and associative.

Parameters

first The beginning of the sequence.

last The end of the sequence.

unary\_op The function to apply to each element of the input sequence.

init The result is initialized to this value.

binary\_op The reduction operation.

Returns

The result of the transformed reduction.

* **copy**:

copy copies elements from the range [first, last) to the range [result, result + (last - first)). That is, it performs the assignments \*result = \*first, \*(result + 1) = \*(first + 1), and so on. Generally, for every integer n from 0 to last - first, copy performs the assignment \*(result + n) = \*(first + n). Unlike std::copy, copy offers no guarantee on order of operation. As a result, calling copy with overlapping source and destination ranges has undefined behavior.

The return value is result + (last - first).

The algorithm's execution is parallelized as determined by exec.

Parameters

exec The execution policy to use for parallelization.

first The beginning of the sequence to copy.

last The end of the sequence to copy.

result The destination sequence.

Returns

The end of the destination sequence.

Source: https://thrust.github.io/

Results:

Sequential:

Text

Description automatically generated

With Thrust:

Text

Description automatically generated

Discussion of the results:

We ran this program sequentially before. Then we ran it with openmp and mpi. It can be easily seen from the results which one is the fastest. But there is something that should not be forgotten here that there are gpu and cpu power differences. For this reason, it is a bit difficult to make an exact performance comparison. But if we look at the time, we can see how much performance we can gain by using the gpu.

What I want to mention next is this: we know that these libraries are actually made for non-developer people (from different fields). As people who have used all 3 techniques, we can say that the most comfortable one is thrust. Thrust was ok for us, but we had problems with a few vertexes. We are not entirely sure of the accuracy of our CSR matrix.

How to compile and requirements:

This features were tried on MacOS(which can support G++).

**g++-12 -O2 -o saxpy thrust.cpp -fopenmp -DTHRUST\_DEVICE\_SYSTEM=THRUST\_DEVICE\_SYSTEM\_OMP -lgomp -I ./thrust;**

g++: G++ is a compiler, not merely a preprocessor. G++ builds object code directly from your C++ program source. There is no intermediate C version of the program. (By contrast, for example, some other implementations use a program that generates a C program from your C++ source.) Avoiding an intermediate C representation of the program means that you get better object code, and better debugging information. The GNU debugger, GDB, works with this information in the object code to give you comprehensive C++ source-level editing capabilities. (<https://gcc.gnu.org/onlinedocs/gcc-3.3.6/gcc/G_002b_002b-and-GCC.html>)

You need to install thrust firstly:

https://github.com/NVIDIA/thrust

g++-12: We used OpenMP 5.0, so we needed to use G++ version of 12. You can check your version just by executing this command “g++ --version”. If your version below, you may need to update your g++.

-fopenmp: The flag is let you code in paralell computing.

-o main.out: It is just to name the output.