Final Project Report

Parallelization of Forward CT Projections

Joel Baby Johnson
ME17B144

ABSTRACT

Nowadays, computed tomography (CT) imaging is being wildly used in varied areas like medical diagnostics, Non-destructive evaluation, Baggage scanners, etc. There is a need to get real-time tomography reconstructions in many of these applications. Iterative reconstruction algorithms are the most commonly used tomography reconstruction algorithms as they are very robust and give good quality results. However, the usage of such algorithms in large-scale or high-resolution imaging systems is still limited, one of the key hurdles being their high computational load.

In order to achieve real-time performance in iterative reconstruction systems, we need faster algorithms. In each iteration of these algorithms, we need to perform forward-projection and back-projection steps, and these are the most time-consuming steps in the reconstruction process. This project aims to make these forward projection algorithms significantly faster through parallelizing to facilitate real-time CT Reconstruction. Also work will be done to parallelize the projection algorithm with MPI to enable work distribution over multiple workstations to make processing even larger datasets feasible.

NOMENCLATURE

The symbols used in the report is specified below.

N_x : No: of volume pixels in x direction

N_y : No: of volume pixels in y direction

N_z : No: of volume pixels in z direction

N_a : No: of detector pixels in x direction

N_b : No: of detector pixels in x direction

SO : Source to origin distance

OD : Origin to detector distance

INTRODUCTION

Iterative reconstruction algorithms offer excellent imaging quality and can be used with varying projection geometries (e.g.: Cone beam, Helical scan, Conveyor belt setup) unlike the conventional analytical reconstruction algorithms. However, these algorithms are much slower because of the number of iterations and large number of calculations required per iteration. In this project work has been done to parallelize the forward projection or X-ray transform step in CT reconstruction algorithm.

Instead of using a ray tracing approach (can be done using vtk [visualization toolkit] package) the project will be looking at a discrete matrix approach as it is easier for parallelization and data handling. The discrete matrix approach for X-ray transform can be formulated in a linear algebra framework as described below

$$AX = P$$

where *X: Volume, P: projection, A: system matrix.*

Each element, a_{ij} (sometimes called weighting coefficient), of the system matrix represents the contribution of pixel j to ray integral i. It can be calculated using the formula given below.

$$a_{i,j} = \frac{\sum_{t=1}^{n} (l_{i,j})_t}{n}$$

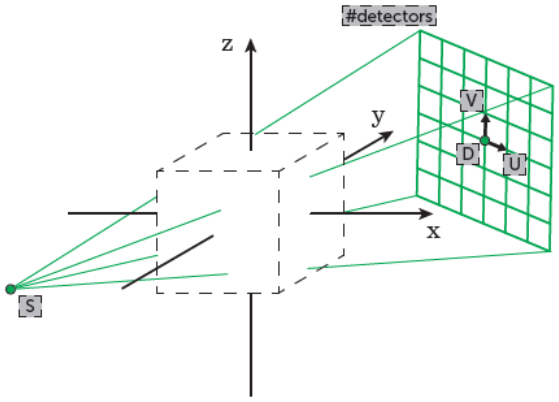
where n is the number of lines per detector and $l_{i,j}$ is the weighting coefficient of each line.

The system matrix is of large dimensions and storing and handling such large matrices are not feasible in most cases and therefore forward projections are computed on the go making use of the discrete matrix idea rather than actual multiplication with system matrix.

There are different algorithms used for realization of discrete X-ray transform. We will be working with line integral model (LIM) proposed by Siddon in 1985.

PROJECTION ALGORITHM

A typical X-ray projection geometry has been shown below. The figure below is a rough illustration of the cone-beam geometry in 3-D volume case where S is the X-ray source, the origin of the co-ordinate system is to be considered at the center of the volume block.



For easier understanding, the X-ray transform using Siddon Algorithm has been explained for a 2-D in this section the same idea can extended for the case of 3-D.

Consider a (N_x , N_y) grid of pixels as our input image whose projection is to be obtained. This image is illuminated with the infinitely narrow rays under the fan-beam CT setting (see Fig (a) below) onto a detector with N_d pixels. For each ray to get the x-ray transform we have to compute the summation of the intersection lengths of the ray with each pixel weighted by the attenuation coefficient of that pixel.

$$y_n = \sum l_{n,i,j} x_{i,j}$$

where x is the attenuation coefficient, $l_{n,ij}$ is the intersection length of the ray n with the pixel (i, j)

The main idea of Siddon's algorithm is the using of the parametric line representation of the rays so that the complexity of computing the intersection lengths of each beam with a 2-D or 3-D domain is still with same as for the 1 dimensional line.

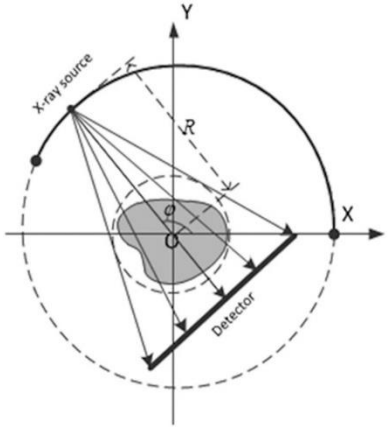


Fig (a). Fan beam Geometry

In case of a 2-D domain with the centre at the origin and unit pixel length for example. The algorithm computes the parametric coordinates of the intersections of the rays with the sets of line

$$\{x = i,\ -Nx/2 \le i \le Nx/2\} \text{ (Columns)}$$

$$\{y = i,\ -Ny/2 \le i \le Ny/2\}, \text{ (Rows)}$$

(which are rows and columns of our grid), sort these intersection coordinates, and then find the non-trivial intersection pixels with the length of intersection for every two consecutive parametric coordinates.

Algorithm in Detail

Consider the intersection of a ray with two end points (x_1, y_1) and (x_2, y_2), with a 2D domain grid of (N_x, N_y) pixels and unit pixel size in both direction and the centre of image is at the origin.

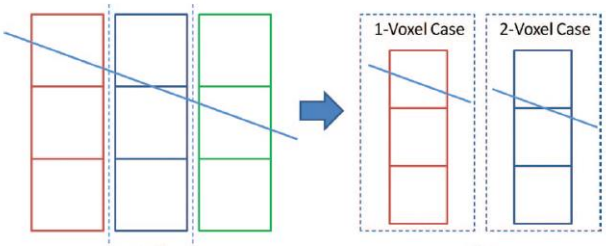
Columns can be given by the equations given below,

$$C_i = \{(x, y) : x = i + 1/2\}, -N_x/2 \le i \le N_x/2 - 1,$$

and similarly, the rows are given by

$$R_j = \{(x, y) : y = j + 1/2\}, -N_y/2 \le j \le N_y/2 - 1.$$

When $|x_2 - x_1| > |y_2 - y_1|$ that is when magnitude of slope of the ray is less than 1, it is better to divide the 2-D grid into the columns. Within each column, the ray intersects non-trivially with at most two pixels .



The indices and the intersection lengths can be determined as shown below.

For the i^{th} column, we have to compute the y-coordinates of the intersections of the ray with two vertical boundaries of this column (Solving line equation in parametric form)

$$y_{i-} = k_y (i - x1) + y1 + N_y/2 \text{ and}$$

$$y_{i+} = k_y (i + 1 - x1) + y1 + N_y/2$$

$$\text{with } k_y = (y2 - y1) / (x2 - x1),$$

and then the points obtain are rounded down to nearest integer to get the corresponding pixel indices.

$$Y_{i-} = \text{floor}\,(y_{i-}) \;\&\; Y_{i+} = \text{floor}\,(y_{i+})$$

If $Yi+ = Yi-$, there is only one intersecting pixel and $1 \le Yi- \le Ny$ then l is given by

$$l = \frac{\sqrt{(x2-x1)^2 + (y2-y1)^2}}{abs(x2-x1)}.Pixelwidth$$

Giving since pixel size of a cell is taken is 1

$$(i, Y_{i-}) : l = \sqrt{1+k_y^2}, \text{ if } Y_{i+} = Y_{i-} \text{ and } 1 \le Y_{i-} \le N_y$$

if Y_{i+} not equal to Y_{i-}, then there are two consecutive intersecting pixels provided $1 \le Y_{i-}, Y_{i+} \le Ny$

$$(i, Y_{i-}) : l = \frac{\max(Y_{i-}, Y_{i+}) - y_{i-}}{y_{i+} - y_{i-}}\sqrt{1+k_y^2},$$

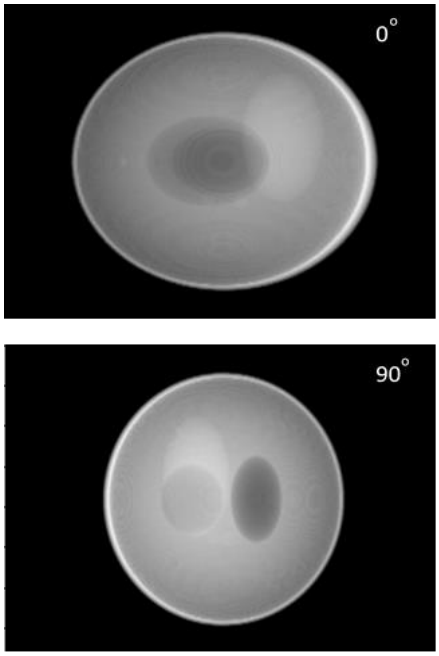
$$\text{if } Y_{i+} \neq Y_{i-} \text{ and } 1 \le Y_{i-} \le N_y$$

$$(i, Y_{i+}) : l = \frac{y_{i+} - \max(Y_{i-}, Y_{i+})}{y_{i+} - y_{i-}}\sqrt{1+k_y^2},$$

$$\text{if } Y_{i+} \neq Y_{i-} \text{ and } 1 \le Y_{i+} \le N_y$$

Similarly, when $|x_2 - x_1| \le |y_2 - y_1|$, we can divide the 2D grid along the rows and proceed in a similar fashion.

The X-ray projections obtained for 0 and 90° for standard Shepp logan volume using the algorithm described above has been shown below



PARALLELIZATION OF ALGORITHM

The projection algorithm has a lot explicitly parallelizable part. For example, the projection to each pixel in the detector is independent of one another. Therefore, they can be easily parallelized. Also, with minor modifications we can easily bring about data parallelism as well for the input volume as well. In this section results of various parallelization methods have been explored to determine the best possible method.

All experimentations on parallelism have been carried out with 3-d Standard Shepp–logan shape. The parameters chosen has been listed below.

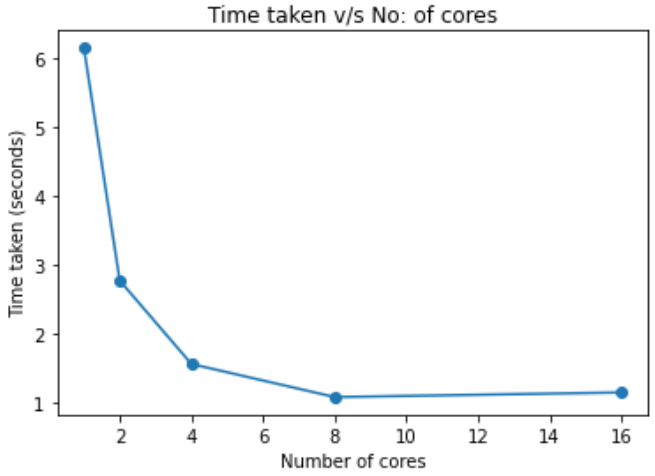
Volume Dimensions: $N_x = 256$, $N_y = 256$, $N_z = 192$

Projection Dimensions: $N_a = 512$, $N_b = 384$

Projection setup: OD = 500, SO = 1000

Open-MP Parallelization

There are about N_a x N_b (196608) pixel projections that can be parallelly run. But only parallelism has been applied on either N_a rows or N_b columns because in practice when the number of threads was increased there was performance decrement after a certain point. This happens because individual time spent in each loop is very small relative to the overall execution time, spawning large numbers of shared memory threads will cause the overhead to overwhelm the execution time required for computing these loops. The time taken versus number of cores graph for experimental case is shown below.



From the graph it is very clear that OpenMP can bring about significant run time reduction in X-ray transform process compared to serial case but since the parallelization potential of the problem is much higher this is not a suitable method for this problem.

OpenAcc Parallelization

Using parallelization methods which are specifically designed for parallelization large number of small problems will be the ideal way to go. Hence attempts for GPU parallelization with OpenAcc was made. As expected, this approach gave much

better performance than the as can be observed from table below.

Method	Time taken (s)
Serial	5.45
OpenMP (n=8, best case)	1.08
OpenAcc	0.34

For given test case where Open-MP gave 5 times speed up we were able to achieve about 16 times speed up with OpenAcc GPU parallelization. But the better news is OpenMP parallelization was at its limit of potential but GPU parallelism can give even better performance for larger size problems because of its capability for even higher parallelization.

Distributed Memory Parallelization with MPI

The GPU parallelization has been proved to be highly helpful but in industrial CT reconstruction applications the memory requirement is much higher. In most cases the entire input volume itself can't even be contained by a single device.

Values for a typical CT reconstruction example is given below to illustrate this point.

Number of projections = 360

Projection dimension = 1024 x 1024 pixels

Volume Dimension =1000 x 1000 x1000 pixels

As can be easily seen the volume data storage itself requires Gigabytes of space. For iterative CT reconstruction algorithms, for each iteration we need these many projections of given resolution. Thus, in most cases there will be a requirement for distributed memory systems to handle processing load.

The process is not explicitly data parallelizable but modifying the physical parameters of the setup a parallelizable model can been obtained. We can obtain projections parallelly for volume slices and then combine them to get total volume projection.

The Volume has been split along z direction (direction perpendicular to the detector's plane). The OD and OS parameters are defined with respect to input volume's origin and not in global origin (Full volume center) so we need to modify those parameters so as to get equivalent physical setup. Adjusting the Detector – Origin (OD) and Source -Origin (SO) distances according to formula given below for each slice will give as same results.

Volume offset = (N_z - (2*Bn-1) * bd) / 2;

OD corrected = OD + Volume offset

SO corrected = SO - Volume offset

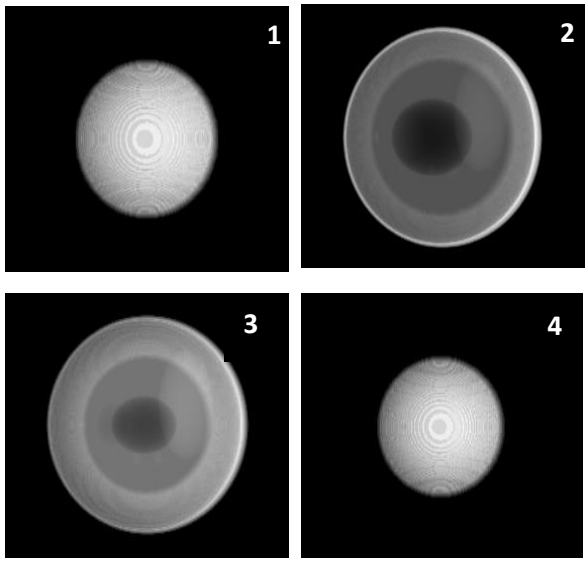
where,

Bn = Block number (Order of slice in original volume)

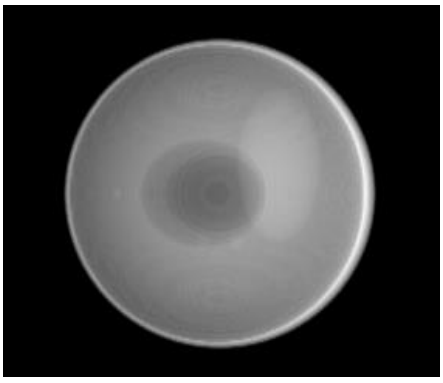
Bd [Block depth] = Volume depth (N_z) / Number of Slices

Data Parallelism has been implemented using MPI for our test case with 4 processes. Each process computed the projection for respective slice parallelly and finally these individual projections from each of the slices of volumes was then be algebraically added to get actual projection of full volume.

The experimentation results showing correctness of this method can be clearly seen from the figures below. (RMSE error of difference with the true volume projection also has been computed to verify the correctness of this approach).



Slice projections sum



MPI parallelization gave a speed up of about 4 for the given test case. Apart from execution time it was able to give significant reduction in data loading time as well. Instead of using a single process loading the entire data and distributing to others, parallel file handling was used. The input data was placed in separate files and individual output was written to separate files and combined later since MPI inter-process communication appeared slower for given test case.

For test purposes the parallel data IO has been facilitated manually but for systems with parallel data handling ability this can be internally dealt with.

CONCLUSION

A combined GPU -MPI parallelization is the answer for real time CT reconstruction problem as it enables fast computation and handling of very large datasets. The potential of parallelization in CT reconstruction applications are enormous. The highly parallelizable nature of the problem gives scope for lot of improvements. Combining all three parallelization approaches discussed in this report advanced parallel algorithm can be developed for parallelizing projection for different angles as well in the reconstruction algorithms.

REFERENCES

1. R. L. Siddon, “Fast calculation of the exact radiological path for a three-dimensional CT array,” Med. Phys. 12, 252–255

2. Willem Jan Palenstijn , Jeroen Bédorf1, “A distributed ASTRA toolbox”

3. Golshan Mahmoudi, Mohammad Reza, “Computationally Efficient System Matrix Calculation Techniques in Computed Tomography Iterative Reconstruction”

4. Hao Gao, “Fast parallel algorithms for the x-ray transform and its adjoint”

5. Matthew D. Jones, Rutao Yao, “Parallel Programming for OSEM Reconstruction with MPI, OpenMP, and Hybrid MPI-OpenMP”