STUDIES OF FORWARD PROJECTION ALGORITHMS

AND IMPLEMENTATION ON PC AND FPGA

By

Minsung Cho

Bachelor of Science - Computer Science
Seoul Digital University
2016

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Engineering - Electrical Engineering

Department of Electrical and Computer Engineering
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2023

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

March 8, 2023

This thesis prepared by

Minsung Cho

entitled

Studies of Forward Projection Algorithms and Implementation on PC and FPGA

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Engineering - Electrical Engineering
Department of Electrical and Computer Engineering

R. Jacob Baker, Ph.D.
Examination Committee Chair

Emma Regentova, Ph.D.
Examination Committee Member

Peter Stubberud, Ph.D.
Examination Committee Member

Mohamed Kaseko, Ph.D.
Graduate College Faculty Representative

Alyssa Crittenden, Ph.D.
Vice Provost for Graduate Education & Dean of the Graduate College

ABSTRACT

This thesis first aims to understand how a part of the computed tomography (CT) algorithm called “forward projector” (FP) works and how it can be accelerated at a hardware level. Two methods of FP are defined and studied: pixel-driven and ray-driven. Both methods fundamentally use the property of line integral and Bresenham’s algorithm. As the result of the study of the two FP methods, the ray-driven algorithm implemented in C++ performed 35% faster than the pixe-driven algorithm implemented in Python.

This thesis implements the ray-driven forward projector algorithm using field-programmable gate array (FPGA), making use of hardware-acceleration techniques. The result shows that the FPGA implementation had a comparable speed advantage compared to the implementation on the personal computer (PC). Even though the FPGA used was outdated and budget-oriented, the chip was able to perform 58% of the performance of a more expensive, modern, and performance-oriented PC. Also, the FPGA implementation performed better in power consumption compared to the PC. Methodologies included the state machine, random access memory (RAM), and universal asynchronous receiver-transmitter (UART). By using the results in this thesis, a CT scanner can be designed to be less expensive, more efficient, faster, and use less power.

ACKNOWLEDGEMENTS

Numerous people have made it possible for me to finish this thesis project. Firstly, I thank my advisory committee chair Dr. R. Jacob Baker for gladly taking the role of the chair and guiding me through the process. I have first met Dr. Baker from his famous “Digital IC” class where he showed true compassion to pass down his expert and relevant knowledge to students to get well-prepared for their job interviews. I also thank the rest of my advisory committee members, Dr. Emma Regentova and Dr. Peter Stubberud. I have met Dr. Regentova from her amazing “Digital System Architecture and Design” class. The class helped me solidify my commitment to study and be an expert in the field of electrical and computer engineering for the rest of my life. I have met Dr. Stubberud from his “Digital Signal Processing” class. His witty ways of teaching dry but important concepts such as convolution and fourier transform magically made them stick with me for an extended period of time. Additionally, I thank Dr. Mohamed Kaseko for being the graduate college representative. His “Engineering Economics” course has taught me a valuable perspective of understanding real-life engineering project cycle consideration.

I thank Imatrex Inc. for providing me with equipment and the knowledge required to complete this thesis project. Most importantly, I thank the company for allowing me to distribute research time and effort to finishing this thesis project while providing a full-time engineering position.

For those who provided me emotional support, I write a sincere gratitude. I share my acknowledgement of my family, my friends, and my labmates. Especially, I thank my girlfriend, Jahzleen, for cheering me up when I was down, motivating me when the progress was slowing, and patiently waiting for me when I had little time for her or when I fell asleep on many dates.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

A modern medical computational (or computerized) tomography (CT) scanner industry increasingly requires larger and faster radiation data to process and compute. Some of the requirements have been met through the power-consuming and expensive graphical processing units (GPU). However, gaining faster computation time with the GPUs comes with the tradeoff of high power consumption and price. This brings up the maintenance and initial price of the scanner, and this makes a life-saving preventative screening process more inaccessible. Hardware-accelerated medical CT scans can solve these issues.

In a CT scan, radiation rays penetrate through a patient to acquire a projected image. These rays rotate around the patient and collect the projected image in multiple angles. Denser parts of the patient, such as bones, absorbs or diffracts the radiation source more than the softer parts of the patient, such as organs. This results in a difference in the intensity of the projected images. CT reconstruction algorithms take these images and reconstruct the cross-sectional image of the patient to detect early development of tumor, internal bleeding, and irregularities inside the body. CT scanner development goes alongside radiation technology to improve medical image diagnostics. CT scanner or image diagnostics have saved numerous lives since its invention by bringing up the percentage of cancer screening to about 20% [1].

A cross sectional image of the patient is mathematically acquired by reconstruction algorithms. Two major reconstruction algorithms are filtered back projection (FBP) and simultaneous algorithmic or iterative reconstruction techniques (SART or SIRT). The forward projection (FP) algorithm is used by these reconstruction algorithms. In FBP, the FP data inversely transform through a back projection algorithm by reconstructed, and in SART, the FP data is repeatedly forward projected and back projected to minimize the weight of the projection

difference to reconstruct the image. Efficient and accurate FP algorithm development is necessary for both reconstruction algorithms to yield an improved reconstruction result. Also, as the radiation source and detector technology progresses, tomographic algorithms such as FBP require more computational power and speed due to high-resolution, high-frequency data acquisition and real-time reconstruction. Therefore, an effort to improve and accelerate the FP algorithm will directly benefit the achievement of such technologies and save more lives.

Recent development of real-time detectors can collect up to 3072 by 3072 of 16 bit pixel data and the speed of up to 30 frames (or 'views') per seconds (fps) [2]. This specification amounts to 4.5Gb of data to transmit, process, and compute per second. CT reconstruction algorithm researchers are mainly focused on optimizing the reconstruction algorithms to parallel processing to process this large data efficiently.. Many of these parallel processing efforts are made using GPUs. Using a GPU is good for elementary and large array vector calculations. For example, multiply-and-accumulate computations found in FBP algorithms perform much faster in parallel processing through FPGA or GPU rather than in CPU. However, there has not been much attempt to accelerate the CT algorithms through the use of FPGA. Generally, in power consumption and cost aspects, FPGAs have more advantage over GPUs for a specified application. The disadvantages of GPUs are that they are heavy on power consumption and, in some cases, slower than application-specific integrated circuits (ASIC) or FPGA hardware implementation due to the lack of optimization.

On top of an effort to improve software FP algorithms, this thesis proposes an effort to implement an FP algorithm on FPGA and highlights the advantages and disadvantages of such an effort versus the software implementations for CPU or GPU. The purpose of the implementation of the FP algorithm on FPGA is to accelerate the previous GPU and CPU based

computation faster with lower power consumption from taking advantage of hardware acceleration. The proposed implementation will reduce the cost of data acquisition and reconstruction by reducing the energy consumption and the device price with better speed and accuracy to contribute to lower price CT device cost that will bring the initial and maintenance cost of scanning down.

2. BACKGROUND

To better understand the topic of this thesis, a bit of background information must be stated about the CT scanner and its components.

2.1 THE CT SCAN GEOMETRY

The CT scanner geometry is divided into two parts: radiation source and the detector. The radiation source is defined as a single point which represents the starting position of the X-rays. The detector consists of a row of pixels that detects radiation emission and converts the intensity from analog to digital. How the radiation source is generated is outside the focus of this study. In a general sense, the radiation source hits the metal target such as tungsten and scatters concentrically from the target point. This scattering beam is collimated, or windowed, and creates a certain angle of exposure to only hit the detector row. When the source emits radiated energy that can pass through the object, the detector collects the intensity of the source after the scatter and absorption by the object in the path. If there's more soft tissue than hard tissues in the beam path, the detector will collect more intensity. This intensity data is the essence of the forward projected data that the FP algorithm aims to simulate.

2.2. THE FIELD OF VIEW (FOV)

An object of interest with unknown density lies in between the source and the detector in the field of view (FOV). The FOV is defined by the angle overlap area of the geometry of the scanner. The resolution and the area of the FOV is defined to limit the computation area based on the distance and length of the detector and the distance of the source. In the CT scanner design

process, a target FOV area and its resolution are set first as a requirement and the surrounding geometry is designed.

Naturally, one could assume that the bigger the FOV the better. However, this is not the case. Depending on the type of scans, the FOV has to trade off between the area and the resolution. In a general full-body scan, the FOV is designed to accommodate large bodies with less resolution density. But in a cardiac scan, the FOV is designed to have a better resolution within a smaller area. In order to have a bigger FOV, either the source and the detector must be further apart, or the detector must be curved. This is because the further the distance between the source and the detector the larger the area that overlaps. This overlap area from the rotation defines the FOV.

2.3. THE DETECTOR

There are many ways the detector can measure the intensity of the X-ray and convert it to a digital signal. One way is to use a scintillator layer to convert the X-ray into light that is digitized by an array of photodiodes. Scintillator is a material that converts X-ray into visible light. Another way is to use a photoconductor layer to directly convert the photons into electric current charge in a capacitor and read out by an array of thin-film transistors (TFT) or solid-state complementary metal-oxide-semiconductor (CMOS) detectors. Recent development of detector technology utilizes photon-counting technique where crystal semiconductors as cathode and anode are used to convert the X-ray directly to electric charge between them and eliminate dead space created from using the scintillator and conversion time from X-ray to light.

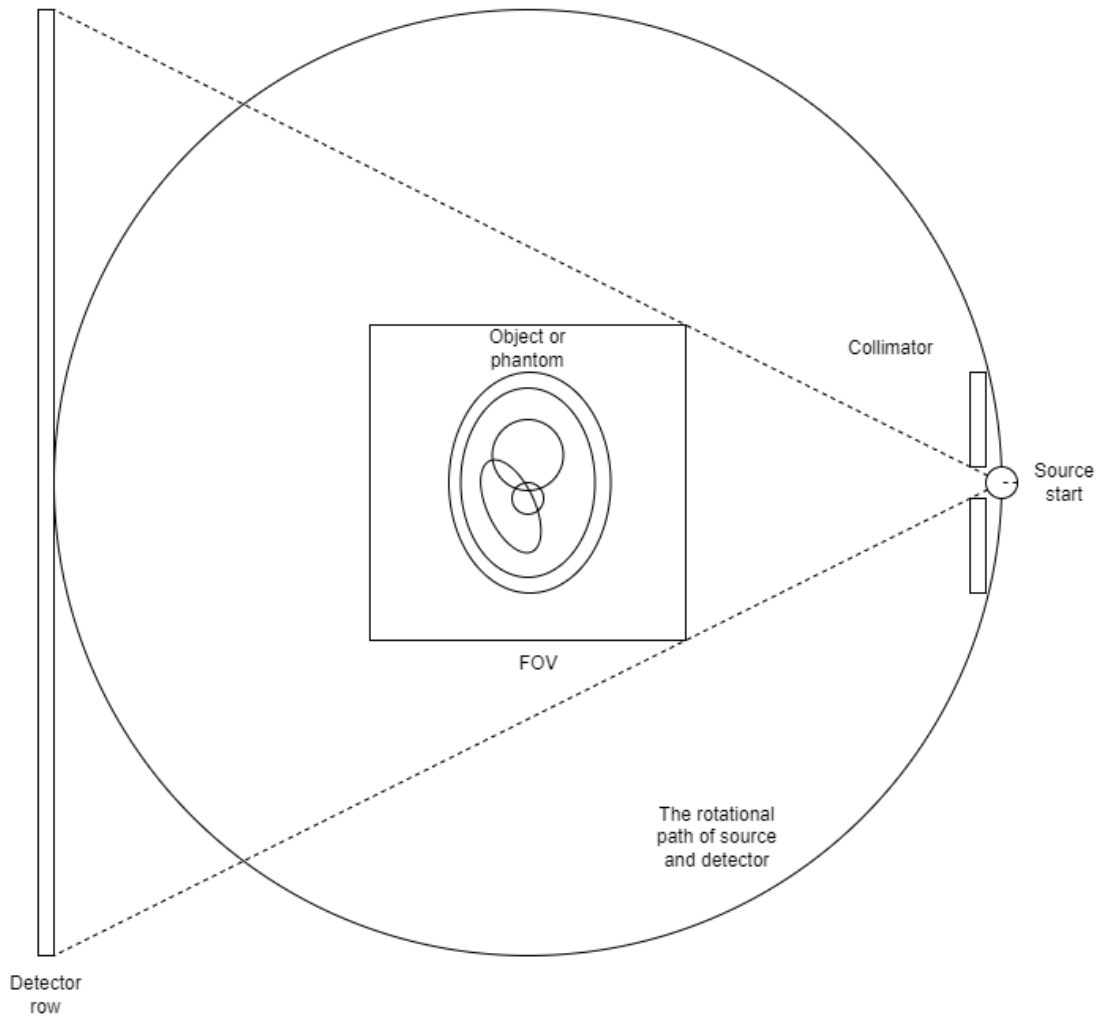

Figure 2.1. Geometric diagram of the CT scan.

The type of detector geometry is typically divided into flat panel or tile (curved array) geometry. Flat panel detectors have a rectangular flat surface such as shown in figure 2.1, and curved array detectors have smaller ‘tiles’ of detectors to create a curved detector surface with a certain radius. Curved detector geometry has the advantage of having a shorter distance between the center for the same size of FOV than the flat panel detector geometry. However, flat panel detectors are much cheaper, computationally less complicated, and easier to develop and

interface. Also, due to the geometric difference, the resulting sinogram image will be slightly different along the outside of the detector rows. The flat panel detector will have the cone-shaped ray whereas the curved detector will have the fan-shape ray.

2.4. CT RECONSTRUCTION ALGORITHMS

In a CT reconstruction, a "view" is referred to as a row or area of detector data collected for a single source point along its rotational path. The most recent CT scanner can rotate in 300 milliseconds. This means that, if the detector's data acquisition interval is 60 fps, then in rotation the CT scan will collect 18 views per rotation. If the average scan time is 15 minutes, then the scanner will rotate 2000 times and will collect 36000 views total. The rotation speed will vary based on the level of detail and noise required for the acquired image. The acquired image is in the form of a sinogram. In the sinogram, the typical convention is that each column represents the detector row and each row represents one view, starting from left to right. The sinogram convention of image alignment is used for the reconstruction because in the FBP the image must be transformed in the order of rotation.

Mathematician Johann Radon's Radon transform introduced in 1917 led to the start of the CT development. Radon transform takes the series of 'projection data,' or sinogram, of lines rotating around a two-dimensional object of an unknown density and returns the density map of the object via inverse-transformation.

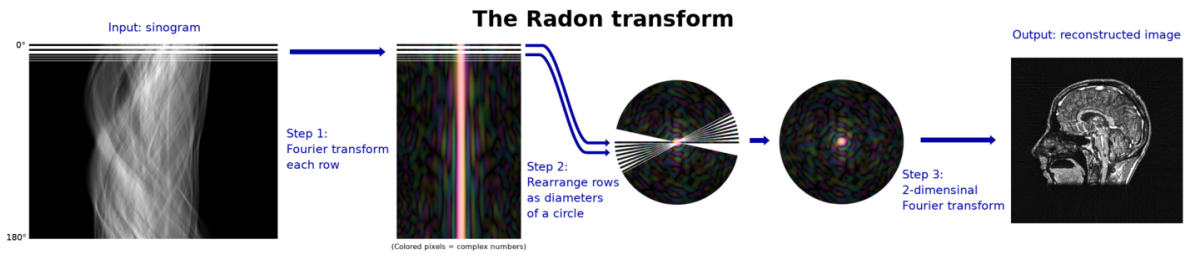

Figure 2.2. Computing the 2-dimensional Radon transform in terms of two Fourier transforms [1]

In 1963, Allan Cormack developed the algebraic reconstruction technique (ART) that uses an iterative reconstruction technique which was adopted in Godfrey Hounsfield's first commercial CT scanner. ART utilizes a different approach than transform-based approach where the unknown cross-sectional data is solved algebraically with the known projection data and ray equations. Ever since their Nobel-awarded invention, Anders Andersen and Avinash Kak improved the ART by making voxel corrections after computing all rays at once (SIRT) and making corrections after each ray (SART).

The FP image refers to a mathematical approximation or simulation of the sinogram data. The FP image is the data acquired by the detectors from the X-ray beam forwardly projected onto them. There are different ways to conceptualize the radiation source geometry to simulate an FP image. The fan-shaped (for 2D) or cone-shaped (for 3D) beams are the most accurate source geometry to the characteristics of CT X-ray physics, compared to the parallel-shaped beam geometry.

2.5. FP ALGORITHMS

The basic theory behind the FP algorithm boils down to taking the line integral of the ray that passes through the object. Ideally, the collected projection data on a detector pixel is the sum of the values that the FOV pixel areas hold that the x-ray passes through. The line integral formula makes it possible to accumulate the infinitesimally small fractions of the points along the ray. First, the property of ray consists of functions of x and y in a parametric equation from the starting point *a* (the source) and to the ending point *b* (the detector).

$$x = h(t),\ y = g(t),\ a \leq t \leq b$$

Because the positron emission x-rays that are used for CT scans travel in a straight line from the source point to the detector point, the function for *x* and *y* can be defined as a linear function. Then the line integral of the ray is denoted by

$$\int_a^b f(h(t), g(t))\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}dt$$

Because the phantom data is binned to fit a discrete FOV with width and height in pixels, the integral should be converted to a summation. Therefore, the FP algorithm is an estimation of the approximate density of a given pixel area based on the weight of the line passing through the area.

The weight can be calculated using two methods: pixel-driven and ray-driven. The pixel-driven method of FP calculates the projection weight from the assumption that the intensity is distributed uniformly on each pixel. The pixel-driven FP algorithm finds the intersection between the ray and the pixel area and accumulates the density weight. The Ray-driven method of FP calculates the weight based on the length of the intersecting line of the given data.

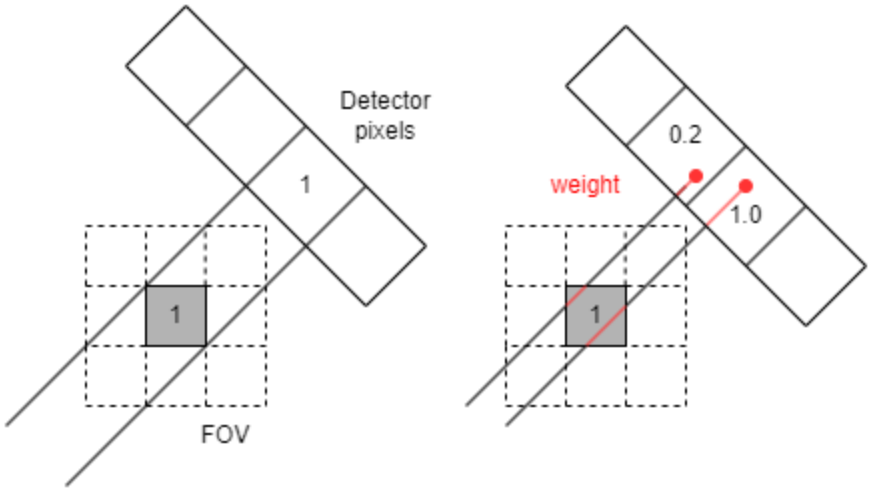

Figure 2.3. From left to right: pixel-driven FP, ray-driven FP.

2.6. PHANTOMS

Various imaging phantoms, or phantoms, are used to tune the beam, adjust the reconstruction parameters, and evaluate the noise performance of the CT scan. Phantoms are specially designed to calibrate for different density of the materials such as water, soft tissue, or hard bones with body parts phantom or to test the quality of the reconstruction with a resolution phantom with thin cylinders and cylindrical holes.

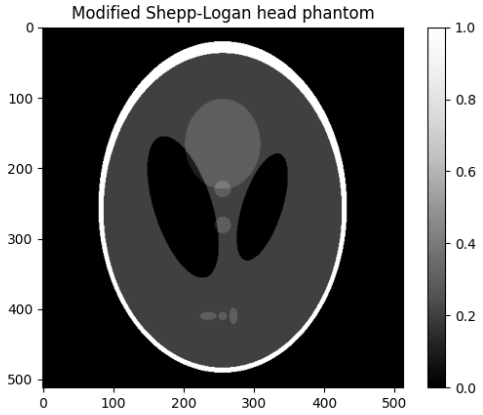

Figure 2.4. Plotted image of a modified Shepp-Logan head phantom.

Shepp-Logan head phantom (SL phantom) as shown in Figure 2.4 is one of the formula-generated phantom data used to test the performance of reconstruction algorithms created by Larry Shepp and Benjamin F. Logan in 1974 [3]. It is composed of ten ellipses defined by functions with center, major axis, minor axis, tilt angle, and the gray level as different parameters. The gray level of the SL phantom represents the linear attenuation coefficient, or the density, of each cylinders. A modified SL phantom has a normalized gray level to improve the contrast. A benefit of the SL phantom is that it can create the cross-sectional image of the head in many different resolutions because it is function-generated.

3. PROJECT SETUP AND ENVIRONMENTS

For this thesis, two FP algorithms were written and tested in three different environments. The pixel-driven FP algorithm is written in Python programming language. The ray-driven FP algorithm is written in C++ and Verilog for both PC and FPGA implementation. For the FPGA, the Xilinx AC701 evaluation kit with Artix 7 XC7A200T was chosen because this chip is budget-friendly. The FPGA firmware was written with development tools such as Vivado, Vitis high level synthesis (HLS), and Vitis integral development environment (IDE). The FPGA is programmable with the free version of Vivado. For the software test environment for Python and C++, AMD Ryzen 9 3900X with 128 GB RAM in Windows 10 Pro was used.

The geometry of the FP algorithm is fixed for all implementations. The detector row is defined as 1000 pixels 1mm wide per pixel. The start of the detector row is 500 mm from the center of the FOV and is on the opposite side to the source point 180 degrees. The source point is located 500 mm from the center of the FOV. There are 1000 views rotating around 180 degrees. The FOV is a grid width of 512 pixels and height of 512 pixels with the side of the square pixel being 1 mm in length. The simulating phantom used in the FOV is a modified SL phantom in 32-bit floating point format. As a result, all sinogram outputs have the width of 1000 pixels and the height of 1000 pixels in 32-bit floating point data. The width of the sinogram output comes from the number of views (1000), and the height of the sinogram output comes from the detector row pixel count (1000).

4. PIXEL-BASED FP ALGORITHM

The pixel-based FP algorithm is implemented in the Python programming language. This algorithm was first written to be the initial proof of concept for the FP algorithm. This algorithm later served as the performance comparison for the ray-driven FP algorithm implementation. With the comparison in the later chapter of the thesis, it highlights the advantage of the ray-driven algorithm over the pixel-based algorithm. It also highlights the different results between the detector geometry. As a major geometric difference, the Python algorithm uses a curved detector geometry, and C++ and FPGA algorithms will use the flat detector geometry.

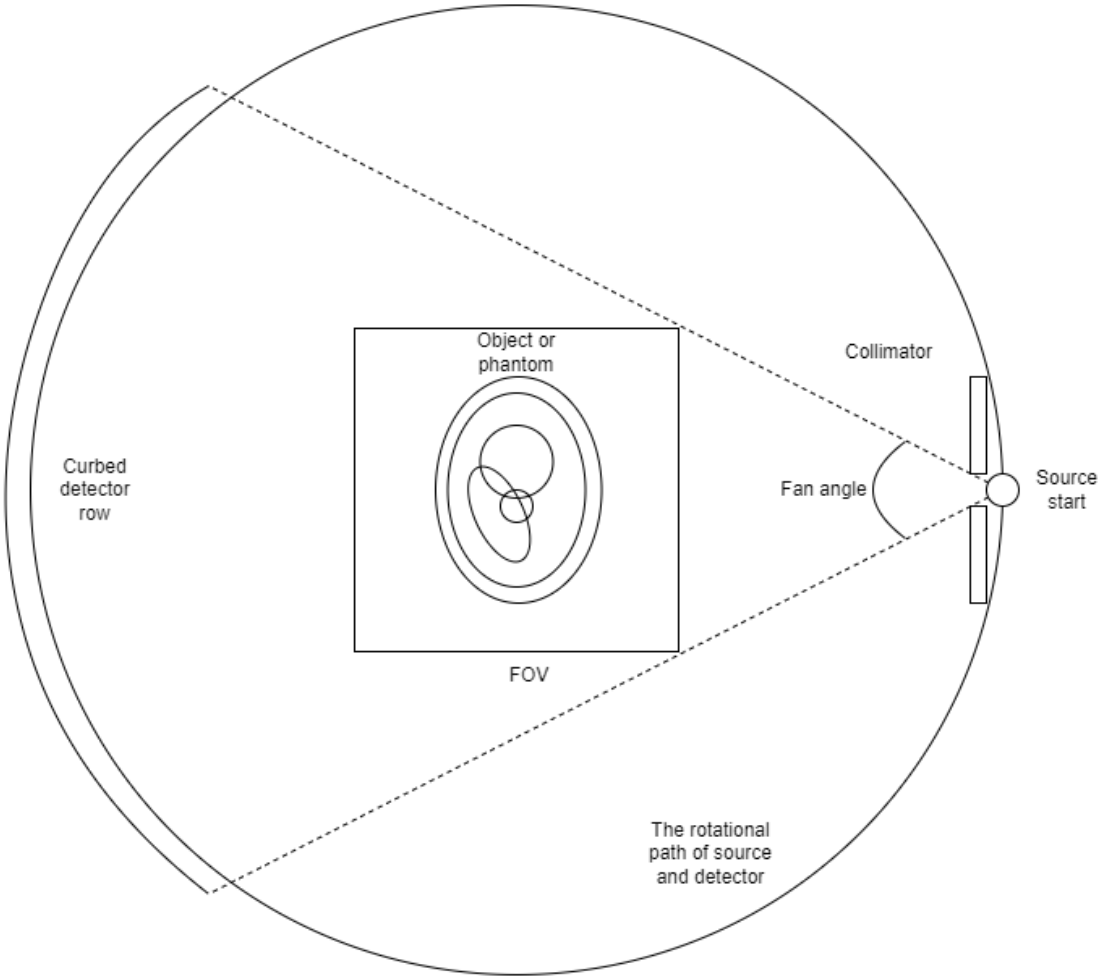

Figure 4.1. Geometric diagram of a curved detector setup.

The geometry of the pixel-based curved detector geometry FP algorithm is as shown in Figure 4.1. above. The algorithm consists of parameters such as fan angle in radians and detector radius. The input parameters are the phantom data (including given width and length), cartesian coordinates of the source positions, and the number of views (a.k.a. the source angle count).

4.1. PRESUPPOSITION OF THE ALGORITHM

The algorithm used to generate the output as shown in this chapter presumes the source radius to be 500mm from the center of FOV, the detector radius to be 500mm from the center of FOV, the fan angle to be $\frac{\pi}{3}$, the number of rays per view to be 1000, and the number of views per rotation to be 1000. The starting source position is at (500, 0), and the scan rotates 360 degrees counter-clockwise. The FOV has the width and length of 512 and 512 pixels, each pixel being 1mm in length. The phantom used to show the result sinogram is the modified SL phantom in 32-bit data in 512 by 512 in size.

4.2. THE METHOD AND DERIVATION OF THE ALGORITHM

The main algorithm used to accumulate the FP is Bresenham's line generation algorithm [4] as shown in Figure 4.2.. This algorithm is widely used and referenced for computer vision applications, such as representing and displaying lines on a discrete and pixelated monitor screen with a continuous line function. The algorithm chooses the pixels that have their center closest to the given line over a fixed increment. For the Python FP algorithm, the selected pixels' data are simply accumulated without the weight factor to acquire the resulting sinogram data. The pixel coordinate convention is set to have the top-left of the phantom data to be the base point; however, this can be changed to accommodate different slopes of lines. For this implementation, only the top-left orientation was considered.

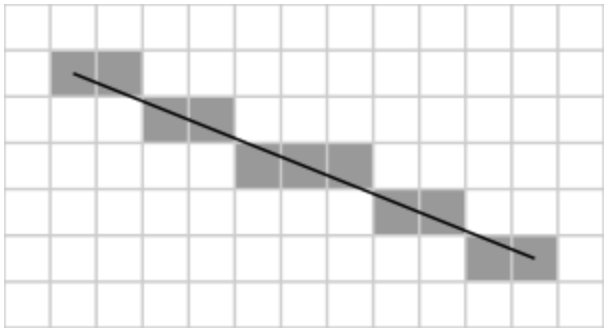

Figure 4.2. A diagram illustrating Bresenham's line algorithm. [5]

At first, the fan angle determines the detector end coordinate prior to executing each view projection. The arctangent of the slope of the view is calculated first with the given detector coordinates and the source coordinates with the x-axis as the base axis. This means that the first view that is queued up for computation has the source position at (500, 0) and the detector center at (-500, 0). The center of the detector is always 180 degrees adjacent to the source point across the center of FOV. If the slope of the ray is negative, then the resulting start angle is a negative angle. However, to traverse the line in the uniform direction, an additional operation of adding π to the angle to rotate the function 180 degrees is done to convert the negative angle to the positive equivalent angle. The starting point of the line traversing is also switched to the detector point, and not the source position of the ray. Based on the fan angle defined, the starting and ending ray coordinates are calculated by subtracting or adding the half of the fan angle to the center of the detector. For example, if the fan angle is set as $\frac{\pi}{3}$ at the first view at 0 degrees from the x-axis, then the start angle would be $-\frac{\pi}{6}+\frac{\pi}{2}$ and the end angle would be $+\frac{\pi}{6}$. The order of rays from start to end is always clockwise. The angular separation between each rays is calculated in equidistance of each other.

Secondly, the source position and detector position from a ray are used to calculate the slope and intercept of the standard point-slope formula. By this, the change in x-direction (Δx) and in y-direction (Δy) are acquired as well as the starting position of the ray. Then, the first pixel to begin the line traversing is determined by rounding down the coordinates of the starting source position. The pixels are traversed equidistantly in the x-direction, and the addition of Δy and the comparison between the center of the pixel determines which pixel is to be included and which is not. With the selected pixel coordinates, corresponding data are accessed from the memory and are accumulated into the corresponding view to the zero-initialized sinogram array. This line traversing is done for all rays in the view, and the next view with the fixed angle increment is done until it fully rotates around the phantom.

4.3. THE RESULT AND CONCLUSION

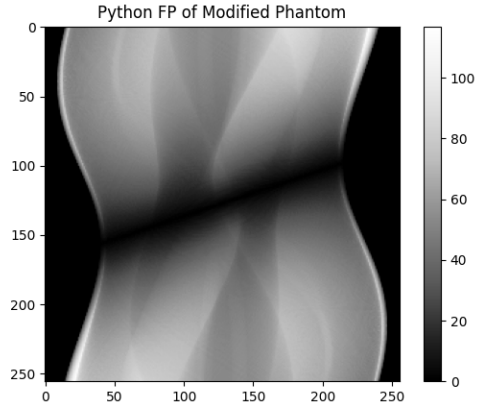

Figure 4.3. Resulting sinogram of a pixel-based FP algorithm.

The resulting sinogram of the algorithm is shown in Figure 4.3. As one can notice, the algorithm has a significant flaw of blurred diagonal line across the middle of the detector rows. This means that there are not enough phantom values accumulated for the region. This is because the algorithm did not consider the vertical orientation of the rays. Because the algorithm is written to increment the pixels in x-direction by one while traverse in y-direction with Δy, there's a limited amount of pixel values vertical to the given x-position.

Based on the disadvantages found from this algorithm, a few improvements can be made; a primary improvement being the inclusion of the vertical rays in the algorithm. An additional feature to be added to the algorithm is the weight factor of the beam overlap onto the pixel location. The pixel-based nature of the algorithm can yield a larger error for the case where the beam that passes through a certain pixel makes a lesser overlap, i.e. the beam that passes a small section of the pixel's corner area. Adding the weight feature based on the beam overlap could reduce the inaccuracy of the FP output.

5. RAY-BASED FP ALGORITHM

The ray-based FP algorithm written in C++ is an improvement of the pixel-based FP algorithm from the previous chapter that is also used for the FPGA implementation. In a nutshell, the ray-based FP algorithm uses the same concept of Bresenham's line algorithm and the pixel increment methodology. However, it has additional features to separate the pixel index calculation between the horizontal and vertical rays. It also adds the weight factor based on how far the offset of the ray's position is to the center of a certain pixel. The algorithm is also optimized for accessing and indexing the phantom data stored in and structured for FPGA implementation. The last difference is that the detector geometry for the ray-base FP algorithm is flat as opposed to the curved detector geometry for the pixel-based FP algorithm. This implies that the flat detector row is perpendicular to the start of the ray position from its center. This also eliminates the need for the fan angle parameter.

5.1. PRESUPPOSITION OF THE ALGORITHM

The algorithm sets some of the geometric constraints as constant, similar to the pixel-based FP algorithm. The FOV size and source positions are identical to the pixel-based FP algorithm. The initial detector *x* position is determined by the detector count and the detector pixel width. The pixel width is set to 1 mm, which is the same for the pixel-based FP algorithm. The detector displacement is equivalent to the detector radius in the pixel-based FP algorithm, but it is the distance between the center of the detector row and the center of FOV. The initial detector x-position is determined by the following formula:

$$D = \frac{-D_c \times D_w}{2} = \frac{-1{,}000 \times 1mm}{2} = -\ 500mm$$

where

D = initial detector x position, D_c = detector count, D_w = detector width

The detector count of 1,000 is also identical. Unlike the fact that the detector count is implicitly determined by the ray count in the pixel-based FP algorithm, the ray-based FP algorithm sets it explicitly. The angle count is also identical to 1,000 per rotation. The source distance is kept constant at 500mm.

The source positions, detector start position, and detector increments are determined by the following formulas:

$$S_x = D_s \times sin(\theta),\ S_y = -D_s \times cos(\theta)$$

$$D_x = D \times cos(\theta) - D_d \times sin(\theta),\ D_y = D \times sin(\theta) + D_d \times cos(\theta)$$

$$\Delta D_x = D_w \times cos(\theta),\ \Delta D_y = D_w \times sin(\theta)$$

where

S_x = source coordinate x, S_y =source coordinate y

D_x = detector start coordinate x, D_y = detector start coordinate y,

ΔD_x = detector increment in x, ΔD_y = detector increment in y,

D_s = source distance, and D_d = detector distance.

The geometry of FOV uses a similar convention to the pixel-based FP algorithm. Each pixel is 1 mm both in length and width. The window of FOV is between (-256, -256) and (256, 256). Therefore, the center of FOV has the coordinate of (0, 0). This convention calls for the need to adjust the pixel coordinate calculations by half the pixel width and length, and this adjust value is calculated as the following formulas:

$$A_x = W_{x,min} + \frac{P_x}{2} = -255.5,\ A_y = W_{y,max} - \frac{P_y}{2} = 255.5$$

where

$$A_x = x\ adjust\ value,\ A_y = y\ adjust\ value,$$

$$W_{x,\,min} = minimum\ x\ window,\ W_{y,\,max} = maximum\ y\ widow,$$

$$and\ P_x = window\ pixel\ width,\ P_y = window\ pixel\ height.$$

5.2. THE METHOD AND DERIVATION OF THE ALGORITHM

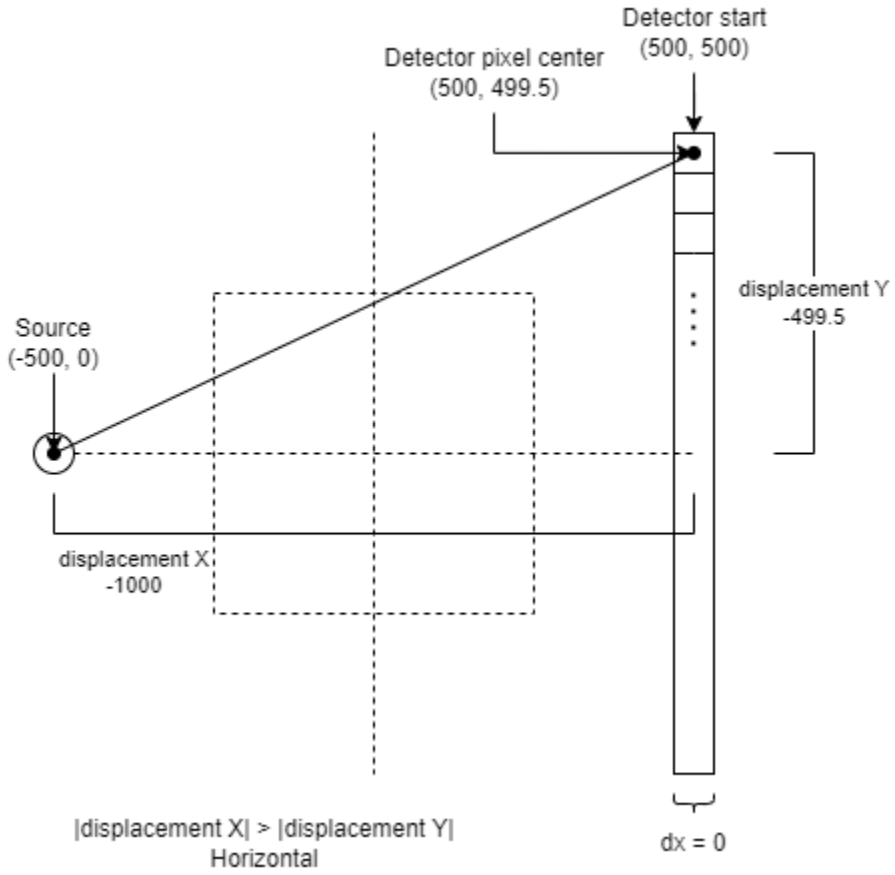

Figure 5.1. Geometric diagram of detector pixel traversal.

First, the algorithm determines a given ray's verticality based on the slope of the ray. This ray is represented by the center coordinate of the detector pixel and the coordinate of the source point. Figure 5.1. illustrates a horizontal ray case. The ray is either vertical or horizontal. The detector pixel center coordinates can be calculated with the formula as follows:

$$D_{ix} = D_x + (i + 0.5) \times \Delta D_x,\ D_{iy} = D_y + (i + 0.5) \times \Delta D_y$$

where

$$D_{ix}, D_{iy} = \text{ detector pixel center coordinates, } i = \text{ detector index}$$

Also, the displacement between the source and the detector pixel is derived as follows:

$$R_x = S_x - D_{ix},\ R_y = S_y - D_{iy}$$

where

$$R_{x,y} = \textit{horizontal and vertical displacement}$$

$$\textit{between the source and the detector pixel center.}$$

The ray is defined to be vertical if the absolute value of the vertical displacement is greater than the absolute value of the horizontal displacement.

In the case of the Figure5.1,

$S_{(x,y)} = (-500, 0),\ D_{(x,y)} = (500, 500),\ \Delta D_{(x,y)} = (0, -1),$

$D_{0x} = 500 + (0 + 0.5) \times 0 = 500,\ D_{0y} = 500 + (0 + 0.5) \times -1 = 499.5,$

$R_x = -500 - 500 = -1000,\ R_y = 0 - 499.5 = -499.5.$

Since the $|R_x| > |R_y|$, the ray is defined as horizontal.

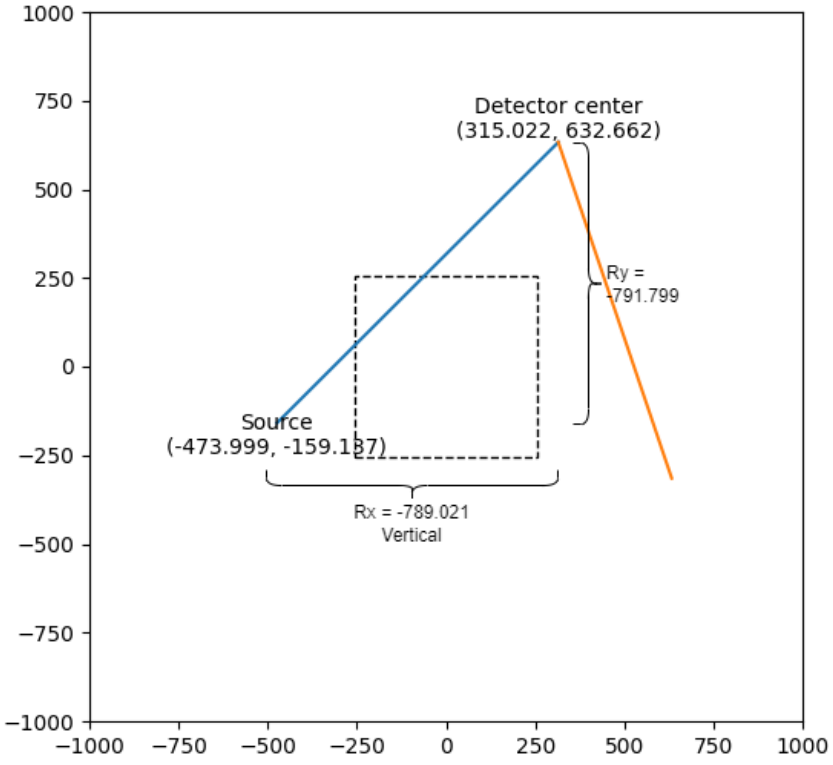

Figure 5.2. Geometric diagram of a ray character calculation.

Vertical rays increment the row by 1 from row 0. The column index is determined by first calculating the initial column value, C, for row 0, and incrementing the change in column value, ΔC and truncating to the closest integer. To derive the C, one must know the change in horizontal direction in respect to the change in vertical direction. This is done by dividing the horizontal displacement between the source and the detector by the vertical displacement.

$$\text{ration of run over rise } = \frac{R_x}{R_y}$$

This ratio is multiplied to the adjusted y coordinate of the given detector coordinate to acquire the weighted displacement in horizontal direction, and it is added to the adjusted x coordinate of the given detector. Finally, the value is normalized with the detector pixel increment. The resulting formula is as follows:

$$C = (D_{ix} + (A_y - D_{iy}) \times \frac{Rx}{Ry} - A_x) \ /\Delta D_x$$

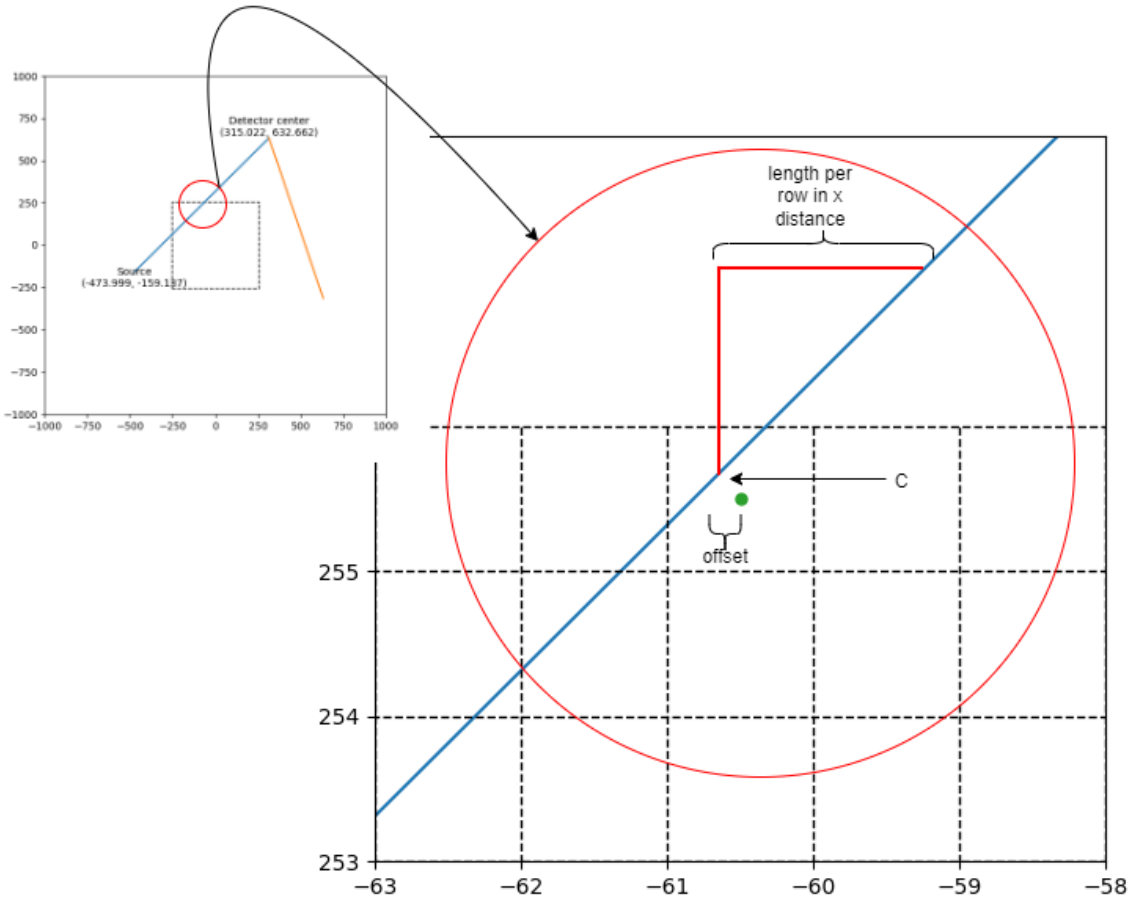

Figure 5.3. Diagram of the pixel offset and weight determination.

The ΔC, since row 0 is defined as the uppermost row, will simply be the negative of the ratio between the horizontal and vertical displacement:

$$\Delta C = -\frac{R_x}{R_y}.$$

The horizontal rays have a similar calculation except that the columns are incremented by 1 from column 0, which is the leftmost column, and the ratio between horizontal and vertical

displacement is inverted. Therefore, the initial row value, R, and the row increment, ΔR, is calculated.

For the vertical case, once the initial column value and the column increment is calculated, then the main loop iterates through each row and finds which column index the ray crosses by truncating the value of C to the nearest integer. If the column is outside the bound of the window, then the loop increments the value of C with ΔC. In a vertical case, there are three cases where the ray will land within the FOV pixel: left to the center, right to the center, or at the center of the pixel area. This is determined by the offset between the real C increment value and the truncated integer column index. This offset is defined as the following:

$$O = C - \overline{C_i}$$

where

$$\overline{C_i} = \text{ truncate } (C + 0.5) \text{ column index..}$$

If this offset is smaller than half of $\frac{R_x}{R_y}$, then the ray is left to the center of the FOV pixel. If the offset is greater, then the ray is right to the center. In other cases, as long as the column index is within the FOV window boundary, the ray is at the center of the FOV pixel area.

The weight calculation depends on the information above since the ray center to the FOV pixel should have greater weight when performing multiply and accumulate than those that are off-center. If the ray is lying left or right to the center of the FOV pixel, then the weight is calculated by comparing the offset with the left (O_S) and right (O_T) bound of the possible offset from the center. They are derived as follows:

$$O_S = \frac{1-Rx/Ry}{2},\ O_T = \frac{1+Rx/Ry}{2}$$

If the offset O is less than $-\ O_S$, then the ray is left to the center, and if it is greater than O_S, then the ray is right to the center.

To derive the weight, the normalized horizontal displacement is calculated as follows:

$$\widehat{C} = \frac{\sqrt{R_x{}^2 + R_y{}^2}}{|R_y|}$$

This displacement vector is used for the following cases to calculate the weights for the left case (W_L) and the right case (W_R):

$$W_L = (O\ + O_T) \times \frac{\widehat{C}}{O_T - O_S},\ W_R = (O\ - O_S) \times \frac{\widehat{C}}{O_T - O_S}.$$

This weight is subtracted from the displacement vector and multiplied to the value of the corresponding FOV pixel intensity with the same row and column indices. The actual weighted length of the crossing ray for the left and right therefore is always less than the maximum ray increment in column per row.

Additionally for the left case, there is a possibility that the ray extends to the pixel left to the current column except for the edge column on the far right. In this case, if the column to the left is within the FOV column bound, then the weight is multiplied to that pixel's intensity value without the displacement vector subtraction. The weight for the center case is simply the normalized horizontal displacement vector itself.

The same algorithm is applied for a horizontal case except that the column is incremented from start to end by 1 while the change in *y* direction per column is calculated and iterated.

5.3. THE RESULT AND CONCLUSION

The output sinogram width is the detector row width and the height is the number of angles in one rotation. The resulting sinogram of the algorithm is as shown in Figure 5.4.

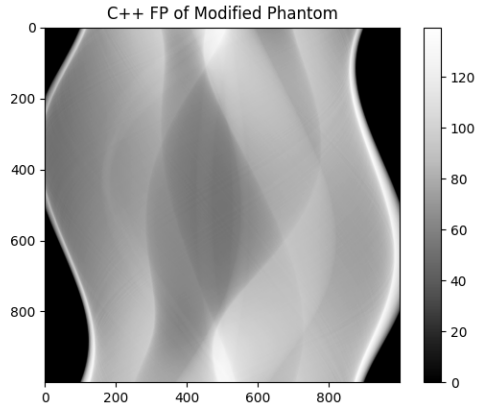

Figure 5.4. Resulting sinogram of a ray-based FP algorithm.

The advantage of the algorithm over the pixel-based FP algorithm written in Python is that the sinogram simulation output has less error due to the weight factor. Another improvement is that the different orientations of ray are handled adequately, and the vertical rays are accumulated well.

Methodology wise, this FP algorithm written in C++ was faster than the pixel-based FP algorithm written in Python. The Python FP algorithm completed the computation in 36.91 seconds and the C++ FP algorithm completed the computation in 23.60 seconds. This is because even though the C++ FP algorithm has the added weight computation, the programming language has overall better performance than Python. Both programs are not optimized by multiprocessing or threading.

The C++ FP algorithm is forked from the open source project, the ASTRA toolbox, developed and maintained by iMinds-Vision Lab, University of Antwerp. The ASTRA toolbox provides a library of a wide range of CT reconstruction algorithms and framework [6].

6. FPGA FP IMPLEMENTATION OF RAY-BASED FP ALGORITHM

The FPGA implementation is designed in two architectures. First is the state machine design pattern where the current state can be dynamically flown into different states. This architecture is used to implement the repetitive step of the FP algorithm. Second is the typical microprocessor design in Harvard or Von Neumann architecture to accommodate the surrounding components such as RAM and communication.

6.1. IMPLEMENTATION STRUCTURE

The structure of the algorithm implemented in the FPGA is similar to the C++ algorithm. The difference between the FPGA implementation and the PC implementation is that the FPGA uses MicroBlaze, a soft microprocessor core, to interface between PC, RAM on the development board, and the hardware-accelerated FP implementation. The main reason for the FP algorithm is implemented in the FPGA this way because there is a great deal of IPs provided to handle a lot of logistical issues that came up while working on this thesis project. One major obstacle that can be overcome by using the microprocessor design is the usage of RAM. By using Xilinx's custom IP module for RAM controller, initializing, writing and reading from RAM is much more convenient in designing the implementation. The RAM controller IP was provided and configured with Vivado memory interface (MIG 7 series).

Another design problem that MicroBlaze solves is the communication between the PC and the FPGA. The implementation uses a universal asynchronous receiver/transmitter (UART) 16550 chip for serial communication. The UART is interconnected with the Advanced eXtensible Interface 4 (AXI4) interconnect module that talks to other AXI modules through

MicroBlaze. A different approach to communication is to use onboard ethernet communication. However, this was not possible for the project due to an IP licensing issue.

Some of the parameters have fixed values for the FPGA implementation similar to the PC implementation, such as the FOV dimensions, detector count and radius, column count and radius, pixel size, and the start and end angle values of 0 to Pi. These parameters are hard-coded, and should be re-synthesized for different geometry.

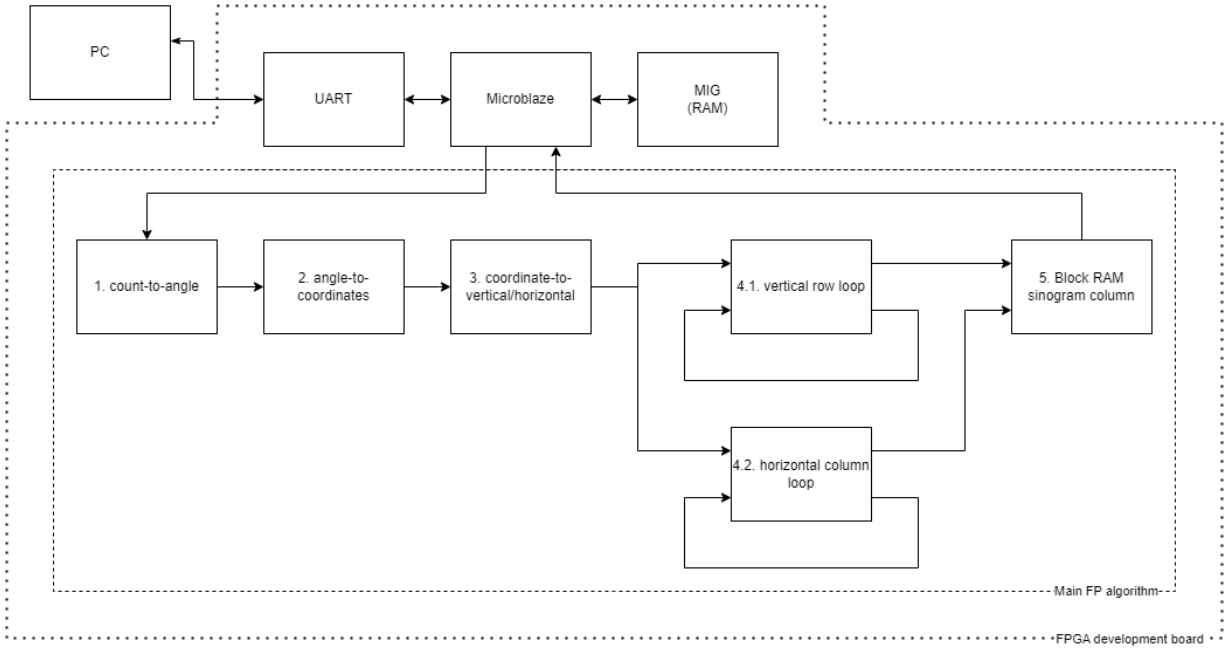

Figure 6.1. Overview diagram of the main FP algorithm for FPGA implementation.

Figure 6.1. shows the top state machine overview of the FPGA implementation. The first step of the algorithm is to convert the integer count value to the corresponding angle from the positive x-axis that is in radians. The second step is to calculate the source coordinate, the

detector start coordinate, and detector *x* and *y* increment based on the angle of the source and the detector. The third step is to determine the characteristics of the ray, whether it is vertical or horizontal. The fourth step is the loop of row and column for each vertical and horizontal ray case and store them in a block RAM. Until the loop reaches the end, the main FP module will read the phantom data from the RAM and accumulate the corresponding data to the block RAM within the state machine. The last step is to store the detector projection row data to the RAM. A result of 1000 data points for each source angle are stored in the MicroBlaze RAM. When all the sinogram columns are written, then the loop emits the 'done' signal through the AXI4 interface to the MicroBlaze processor. Generally, the modules generated by Vitis HLS tool follow the communication protocol similar to AXI4 protocol. This protocol is universally used and followed by many microprocessor systems and modules. The protocol is active when reset is LOW. When start is HIGH, the module starts. When the module starts, idle is driven LOW until the operation is done. When the data is ready and done, both ready and done will be HIGH for one clock period. After that, when the module is not active, idle will go HIGH. Most of the sub-modules are created via Vitis HLS.

6.2. THE PHANTOM AND THE DATA TYPE

CT detectors convert the analog intensity data into digital data. This value represents the electric energy that passed through the scintillator to the detector in an analog capacitive energy that gets converted by 16-bit analog-to-digital (ADC) converters. To simulate the projection image resulting from the actual scan, the sinogram data should be an unsigned 16-bit integer data. Most reconstruction algorithms convert the detected image data to attenuation coefficient in a floating point format such as 32-bit floating point value. This is done with normalizing the

attenuation of water molecules using a water calibration phantom. Especially for GPU reconstruction using Nvidia's CUDA parallel computing programming, the data is converted to 64-bit floating point to support the architecture. This FPGA implementation is done in 32-bit floating-point data type. The Institute of Electrical and Electronics Engineers Standard for Floating-Point Arithmetic (IEEE 754) format is used for the floating point format. It includes the sign, the exponent, and the mantissa. The exponent takes 8 bits and the mantissa takes the 23 bits with 1 sign bit that makes the 32 bits.

The integer and the decimal value are first separately converted into their binary representations. For example, the number 85.125 has the integer 85, which is 1010101 in binary, and has the decimal 0.125, which is 0.001, and it has the binary representation of 1010101.001. The signed bit is 0 since the number is positive. In scientific notation, the number is 1.010101001×2^6. The '6' is added to the single precision bias 127 to make 133, which is 10000101 in binary. This will be the 8 bits exponent. The exponent '010101001' is stored to the mantissa with added zeros for the rest bits. The final IEEE 754 single precision of the number is 0 10000101 01010100100000000000000 or 0x42AA4000.

The Shepp-Logan (SL) head phantom data is in a double precision (64-bit) floating point from 0 to 1 that resembles the attenuation coefficient of soft and hard tissues. The attenuation coefficient is a value to express how easily the material can be penetrated by the X-ray. As another widely-used equivalent unit conversion, there is the Hounsfield unit where the water is represented as 0, the air is represented as -1000, and the bone is represented as greater than 1000. The modified Shepp-Logan head has a higher contrasted range of values for better visibility. The phantom consists of multiple defined ellipses that represent different regions of the brain. The frequent values of the SL phantom can be found in Table 6.1. Because there are only a handful of

ellipses, the variation of the pixel value is not limited to the values listed on the table below. Since all of the values point to much shorter and simpler numbers, the loss in accuracy can be somewhat forgiven when converted from double precision to single precision.

	SL phantom values	Modified SL phantom values	Counts	Percentage
1	0.0	0.0	131968	50.34%
2	1.734723475976807e-17	-5.551115123125783e-17	20080	7.66%
3	0.010000000000000018	0.09999999999999995	361	0.14%
4	0.020000000000000018	0.19999999999999996	86683	33.07%
5	0.03000000000000002	0.29999999999999993	11396	4.35%
6	0.04000000000000002	0.3999999999999999	200	0.08%
7	1.0	1.0	11456	4.37%

Table 6.1. The SL phantom vs modified SL phantom value, count, and percentage comparison

6.3. MEMORY REQUIREMENT

For the implementation in FPGA, the phantom data is stored in the RAM. The data is written to RAM on the FPGA development board from the PC via universal asynchronous receiver-transmitter protocol (UART). The sinogram data is written to RAM when the computation is done. The algorithm will utilize block ram to store each column of sinogram data and is written in RAM after each ray position. Table 6.2. below states the memory requirement for the data.

Name	data type	size
Phantom data	32-bit single precision floating point	32 bits * 512 * 512 / 8= 2MB
Sinogram data	32-bit single precision floating point	32 bits * 1000 * 1000 / 8 = 4MB
One view data	32-bit single precision floating point	32 bits * 1000 / 8 = 4KB

Table 6.2. Data types and sizes by category.

The block RAM storage for the chip is not sufficient to store both phantom and the resulting sinogram data. Therefore, the external RAM is required in this design. If the resolution of the sinogram or the phantom is smaller, it is possible to reduce the design by only using the block RAM.

The phantom and the sinogram are both flattened when stored. The order of the data stored goes from row to column. For example, the data is stored in the following order: row 0 column 0, row 0 column 1, … row 0 column 511, and row 1 column 0.

6.4. THE ALGORITHM IMPLEMENTATION MODULES

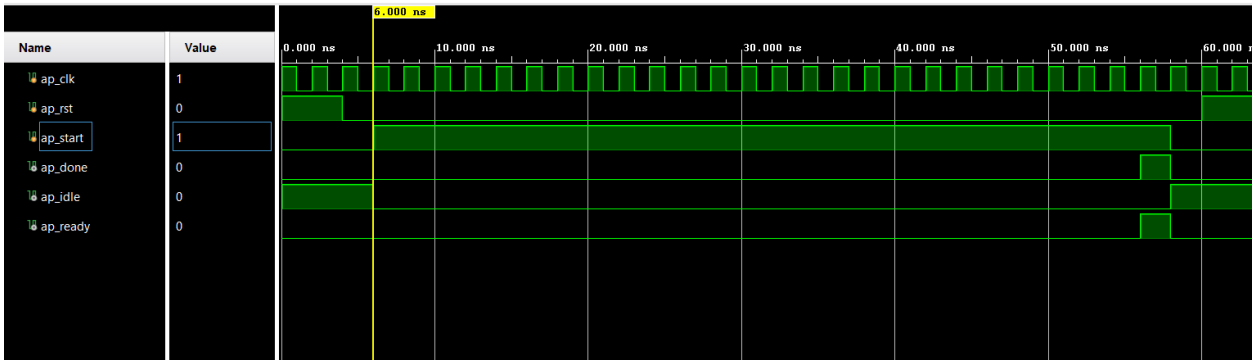

Figure 6.2. Timing diagram of an example AXI4 protocol.

An example RTL timing of the modules are shown in the timing diagram in Figure 6.2. The reset (ap_rst) is active LOW. The design starts when ap_start is asserted HIGH. ap_idle is driven LOW when this happens. The data input and output should be ready when the ap_start is asserted. When the operation is done, ap_ready and ap_done are signaled HIGH. Then the ap_start is driven LOW and the state is back to idle.

6.4.1. COUNT-TO-ANGLE (ANGLE) MODULE

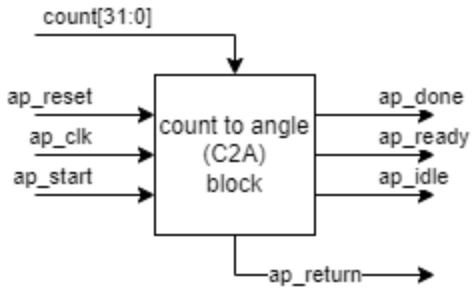

Figure 6.3. Example module diagram from the state machine.

The angle module converts the integer count to a single precision floating point angle value of the source and the detector in radians. The typical input and output signals for the communication protocol is shown in Figure 6.3. The module has inputs of 10ns clock (ap_clk), reset (ap_reset), start enable (ap_start), and 32 bit unsigned integer count, and has outputs of done signal (ap_done), idle signal (ap_idle), data ready signal (ap_ready), and 32 bit floating point angle value (angle). In the unit test, the conversion operation takes 9 cycles of 10ns rising edge clock (90ns). The operation time the module adds to the total algorithm is 90ns * 1000, which is 90µs. An example of an output can be shown in Figure 6.4.

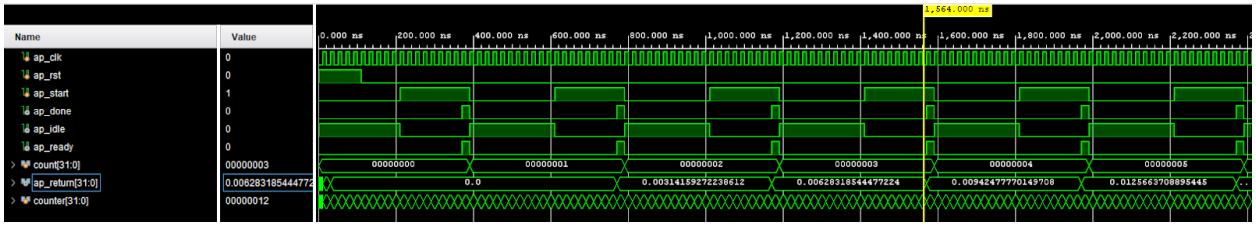

Figure 6.4. Timing diagram of the angle module.

The angle module includes operations such as floating point multiplication. The module is written and generated using the HLS tool.

6.4.2. ANGLE-TO-COORDINATES (COORDINATES) MODULE

The coordinates module calculates the source coordinates, detector start coordinates, and detector coordinate displacement, as well as the same operation signals as the count-to-angle module, such as ap_reset. The operation begins when the ap_start signal goes high. This signal is pulled up at the state where the count-to-angle operation for the count is done. In the unit test, the operation takes 30 cycles of 10ns rising edge clock (300ns). The operation time the module adds to the total algorithm is 300ns * 1000, which is 300µs.

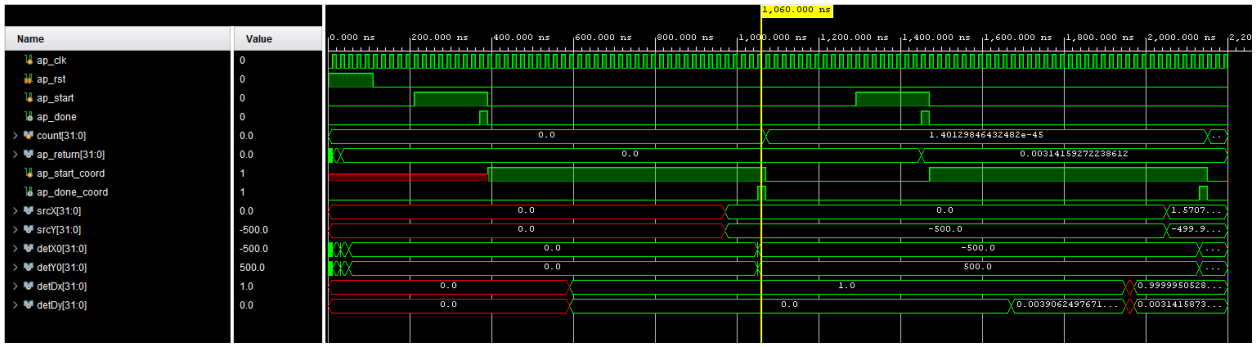

Figure 6.5. Timing diagram of the coordinates module.

The coordinates module includes operations such as floating point sine and cosine, multiplication, addition and subtraction. An example operation can be shown in Figure 6.5. For sine and cosine operation, Coordinate Rotation DIgital Computer (CORDIC) technique, developed by Jack Volder, was used to efficiently calculate the output ratios [7]. An example of CORDIC technique is illustrated in Figure 6.6. CORDIC is an iterative algorithm to adjust the rotation of the angle based from the positive *y* axis in a loop to arrive at an accurate trigonometric value. The number of iterations is handled by the CORDIC IP. In essence, there's an angle of a ray to estimate the value of in radians, and it takes additions and shifts from an initial angle to iterate and guess. In detail, it applies the rotation transformation matrix in Equation 6.1. on the unit i-direction vector.

(Eq. 6.1.)

$$x' = x\cos\theta - y\sin\theta = \cos\theta(x - y\tan\theta)$$

$$y' = y\cos\theta + x\sin\theta = \cos\theta(y + x\tan\theta)$$

The θ is iteratively rotated clockwise or counter-clockwise. After constraining the θ, the rotated matrix would be as below where n is the number of iterations. The θ is from an angle from a look-up-table in the IP. The constrained terms are a simple right shift operation.

$$x_{n+1} = \cos\theta(x_n - y_n\tan\theta) = \cos\theta(x_n - y_n(2^{-n}))$$

$$y_{n+1} = \cos\theta(y_n + x_n\tan\theta) = \cos\theta(y_n + x_n(2^{-n}))$$

where

$$\theta = \tan^{-1}(2^{-n})$$

For the cos θ, it is replaced by a predetermined value based on the number of iterations, and the starting angle is set to the angle of interest.

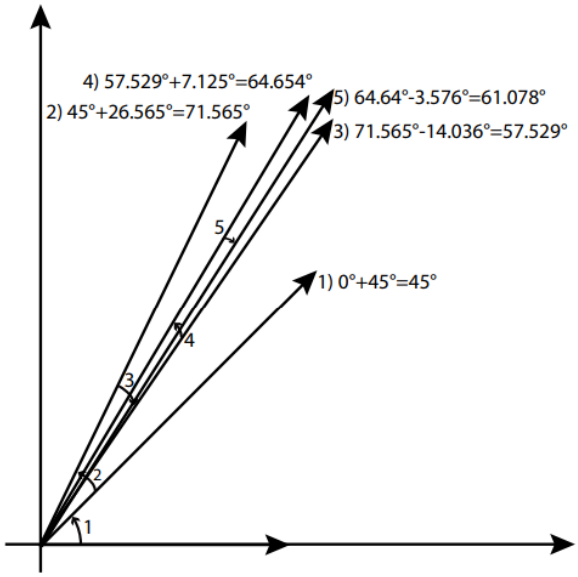

Figure 6.6. Example calculation of $cos60^{\circ}$ and $sin60^{\circ}$ using the CORDIC algorithm. [8]

6.4.3. THE COORDINATE-TO-ORIENTATION (ORIENTATION) MODULE

The orientation module takes the source positions, detector positions, and the detector increment values and outputs whether the ray of the angle is horizontal or vertical. This is done by calculating the displacement between the X and Y of the source and detector of a certain detector index. If the X displacement is greater than the Y displacement, then the ray is defined as horizontal.

In the module, the floating point displacement x and y are converted to an absolute value. This is very simple since only the sign bit can be pulled to zero to convert all values to positive for IEEE 754 format.

The detector center displacement (Dx, Dy) and the total displacement between the source and the detector center (Rx, Ry) are also passed onto the next module. The orientation module takes 26 cycles of 100MHz clock in the unit test, or 56ns, and the total number of operation in the algorithm is 1,000 * 1,000 = 1,000,000 times (angle index times detector pixels). This totals to 56 ms of operation, excluding the data ready clock cycles and reset cycles. The data structure of the return data from this module looks as Figure 6.7.

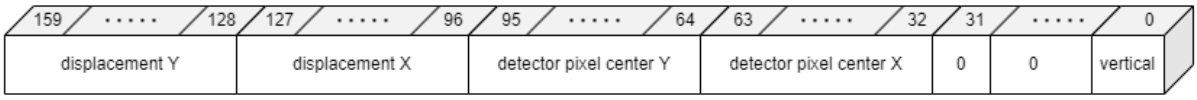

Figure 6.7. The data format of the five outputs from the orientation module.

6.4.4. THE LOOP CALCULATION MODULES

If the ray being calculated is vertical, then the algorithm loops the rows and increments the column along the ray. If the ray is horizontal, then the algorithm loops the columns and increments the column along the ray. Before the loop with row and column decrement, the algorithm calculates the variables described in the ray-based FP algorithm chapter. The module saves the row and column indices and the corresponding weights to the block RAM module, which is the next state. In the unit test, the loop algorithm takes 1.4μs per pixel. For the main algorithm implementation, it took 40.93 seconds. An example of this operation is found in Figure 6.8.

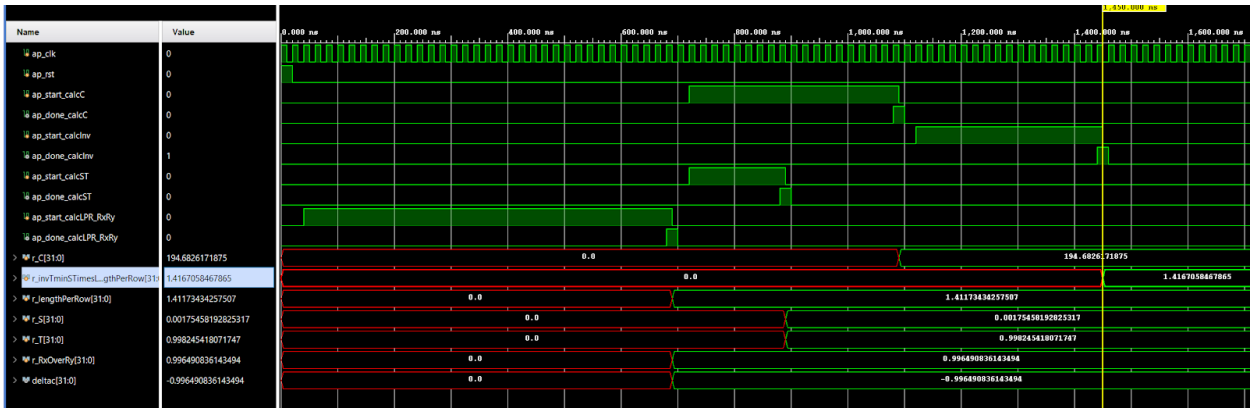

Figure 6.8. Timing diagram of the loop module.

6.4.5. WRITE SINOGRAM COLUMN TO RAM

The block RAM module has a stacked data of the phantom data indices and the weights. It reads the phantom data from RAM through the Microblaze and applies the weight and stores the weighted detected values to RAM. When the loop count is at the maximum count, then the next data is not queued for the main FP module and the sinogram data can be read from the PC through UART.

The file transfer and status communication are done on the PC side using TeraTerm. The soft application was written in Vitis IDE. The language used to build the application was in C++. The main algorithm state machine in RTL was designed in the Vivado block design tool using the IP integrator. The main state machine was wrapped in an AXI4 to communicate with the MicroBlaze through AXI interconnect. The modules used in the main algorithm are written in C++ form Vitis HLS and were exported as an IP to be used in Vivado.

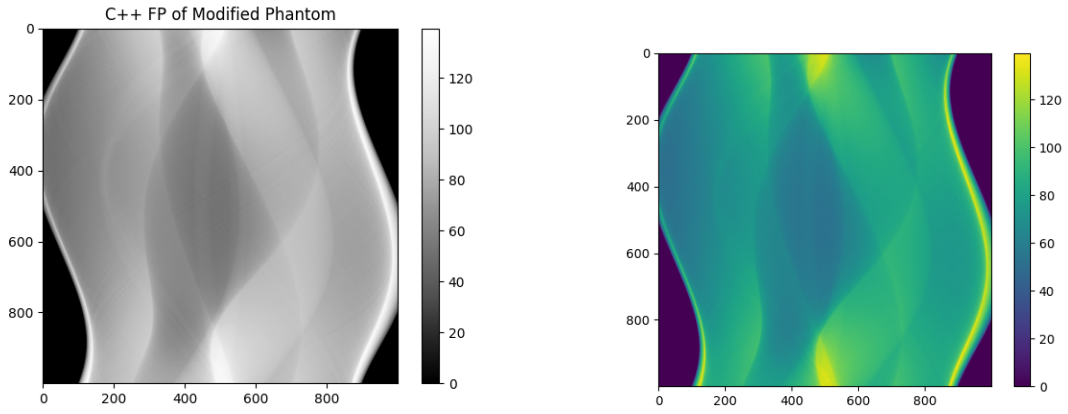

Figure 6.9. The comparison of sinogram from left to right: C++ and FPGA

The resulting sinogram of the algorithm is shown in Figure 6.9. The FPGA algorithm generated a sinogram close in shape and intensity to the sinogram from the C++ algorithm. The major difference in the values comes from the difference in the CORDIC IP's trigonometric estimation. However, the CORDIC algorithm's output is also accurate to the true values of sine and cosine. The minor difference in values come from floating point calculations within the FPGA, which is handled by the Vitis HLS when the IP is created. This can create rounding errors. Also the floating point arithmetics in the IPs result in slightly different decimal points. Regardless, the resulting sinogram does not vary much from it since the added values and its weights are fractional. On top of that, the output of the algorithm has been verified with a golden value generated by the C++ algorithm, and the average percent difference measured to be less than 1%. It shows that the FPGA implementation of the FP algorithm is possible.

In terms of the performance of the speed, the algorithm implementation by itself took 41 seconds. However, the bottleneck in speed comes from reading/writing from the external RAM through the UART protocol. The protocol uses the outdated USB 2.0 bus interface which can only transfer data at 480 Mbps. Theoretically, the transfer speed is not significant: 0.02 s for the phantom data transfer and 0.06 s for the sinogram data transfer. However, the baud rate of 9600 was set for UART, and the entire data transfer took more than an hour to transfer the phantom data and sinogram data between the PC. This is a fatal design flaw of the project that can be improved significantly for the production application.

6.6. FUTURE IMPROVEMENTS

As the section above mentioned, there are clear disadvantages in implementing the FP algorithm in such a way. First, each ray traversing algorithm was done sequentially. This can be improved to handle multiple ray traversing in parallel. Even though each step of the ray traversing algorithm was optimized to fit more steps in a loop, each ray calculation still goes through increments in a sequential pattern. In the future improvement, the ray traversing algorithm should be parallelized to handle multiple ray traversing at a time. This would mean that multiple cores of the ray traversing algorithm will calculate a stack of data coordinates to be stored in a RAM to be handled more efficiently.

Second, within the ray traversing algorithm, finding the overlaid pixel coordinates with Bresenham's line equation can be optimized. The current design first calculates the pixel coordinate, hangs while the data is read, applies the weight, and accumulates into the ray sum. The ray traversing algorithm can perform each step in a separate module for optimization. A proposed change in the architecture would reflect the steps as follows:

1. The first module calculates every ray's overlying pixel coordinates and stores them and the corresponding weights in a sorted stack.
2. The next module reads the data of the coordinates and stores the weighted data into a stack.
3. The final module accumulates the weighted data stack.

This will reduce the occurrence of one process from holding up the rest of the process in the ray traversing algorithm.

Lastly, the data transfer between the PC and the FPGA system should be optimized. It was observed that the bottleneck of the implementation came from the transfer. The first approach is to replace the microprocessor design to a more simplified RAM, communication, and the state machine controller. Because of the inability to control RAM without a proprietary IP through the MicroBlaze, the solution used for this project was not the most lightweight and optimized choice. Next, the UART communication should be either replaced by an external storage device such as an SD card or a faster ethernet communication. Another potential method is to establish the Xilinx direct memory access (XDMA) subsystem to bypass the CPU resources and have a mapped memory between the PC and the FPGA board memory. This will greatly reduce the data transfer time between the FPGA and the PC. And this will also bring better control of the system by setting register maps for variables and different modes that can be directly controlled from the PC.

7. CONCLUSION

First in this study, the different types of FP algorithms are explored and implemented. The first iteration of the pixel-driven FP algorithm has the advantage of having a more simple algorithm composed of shift-and-add operations. It has the disadvantages of poorly handling vertical rays, lower accuracy of the projection data due to the inconsideration of line weights, and lower performance from the programming language.

Some improvements have been made to the ray-based FP algorithm. One of the improvements over the pixel-driven FP algorithm is that the ray-driven FP algorithm handles the vertical ray cases well. Another improvement is that the inclusion of the line weight algorithm achieves better accuracy in result. The other improvement is a slight performance boost and optimization from lowering the level of the programming language from Python to C++. The disadvantage of the ray-based FP algorithm was that it was not parallelized by nature. This invoked the question of whether parallelizing the algorithm via GPU or FPGA would improve the performance of the algorithm.

With the question above, the algorithms are implemented for the PC and the FPGA. It has been found that there are clear advantages and disadvantages of implementing the FP algorithm on the FPGA over the PC. As the main advantage, the implemented algorithm itself performed relatively better than the similar PC algorithms. The result of the FPGA algorithm is relatively fast compared to 36.91 s taken in Python and 23.60 s taken in C++ considering the fact that the chip, Artix 7 series, is based on the 28 nm process released in 2010 compared to the CPU used, AMD Ryzen 9, that is based on the 7nm process released in 2019. Therefore, it is reasonable to conclude that the FPGA implementation of the FP algorithm brings a performance edge.

Another implied advantage is in the price and power consumption. The below chart shows the general advantage of price and average power consumption between the FPGA, GPU, and CPU, from left to right. In a power-consumptious project such as a medical CT scanner, it might not seem like a great deal to add a couple of thousand watts for computing reconstruction and data acquisitions. However, reduction in power consumption is a great way to make the product more carbon friendly.

Name	XC7A200T	AMD Ryzen 9 3900X	NVIDIA RTX 3080 ti
Price	$337.40	$379.99	$1,000 >
Average power	3.52W	86.7W	384W

Table 7.1. Price and power comparison of different implementations as of 2022/09/12.

However, there remains a question whether it is objectively better to implement the FP algorithm in FPGA than in CPU or GPU. Considering the disadvantages from the previous chapter, the performance advantage was far outweighed by the lack of consideration in optimizing the solution for the surrounding system architecture. Nevertheless, the proposed improvements mentioned in the same chapter should be attempted to provide more evidence to the argument that the FP algorithm is better to be implemented in the FPGA hardware rather than in CPU or GPU applications. Without the much needed improvements, it is better to implement the algorithm in PC.

8. FUTURE WORK AND IMPROVEMENT

Beyond the studies of the FP algorithm and the optimization effort, a back projector and reconstruction algorithm should be studied and optimized. The back projector or reconstruction algorithms use various mathematical operations that can be optimized better in parallel than the FP algorithm in its mostly sequential nature. After all, the purpose of the FP algorithm lies heavily in the utilization in the back projector and the reconstruction algorithm.

REFERENCES

[1] “Cancer Facts & Figures 2022.” American Cancer Society. https://www.cancer.org/content/dam/cancer-org/research/cancer-facts-and-statistics/annual-cancer-facts-and-figures/2022/2022-cancer-facts-and-figures.pdf

[2] P. Selinger. “Illustration of how to compute the 2-dimensional Radar transform in terms of several Fourier transforms.” Own work. 11 September 2018. https://commons.wikimedia.org/wiki/File:Radon_transform_via_Fourier_transform.png

[3] iRay Jupi 1717X technical parameters. https://www.iraygroup.com/site/productDetailMore/176?lang=EN.

[4] L. A. Shepp and B. F. Logan. “The Fourier Reconstruction of a Head Section”. Bell Laboratories. IEEE Transactions on Nuclear Science, Vol.NS-21, June 1974.

[5] Bresenham, J. E. “Algorithm for computer control of a digital plotter”. IBM Systems Journal. 4 (1):25-30. 28 May 2008.

[6] Crotalus horridus. A diagram illustrating Bresenham’s line algorithm. Wikipedia. 17 December 2007. https://commons.wikimedia.org/wiki/File:Bresenham.svg

[7] W. van Aarle, W. J. Palenstijn, J. Cant, E. Janssens, F. Bleichrodt, A. Dabravolski, J. De Beenhouwer, K. J. Batenburg, and J. Sijbers, “Fast and Flexible X-ray Tomography Using the ASTRA Toolbox”, Optics Express, 24(22), 25129-25147. 2016. http://dx.doi.org/10.1364/OE.24.025129. https://github.com/astra-toolbox/astra-toolbox.

[8] R. Kastner, J. Matai, and S. Neuendorffer. “Parallel Programming for FPGA”. ArXiv e-prints. https://arxiv.org/abs/1805.03648. May 2018. https://github.com/KastnerRG/pp4fpgas.

[9] R. Kastner, J. Matai, and S. Neuendorffer. Figure 3.2. “Parallel Programming for FPGA”.

CURRICULUM VITAE

Minsung Cho
(702) 717-0666
ms98.cho@gmail.com
1965 Verbania Dr. Las Vegas, NV 89134

Education

University of Nevada, Las Vegas, Electrical Engineering MS & BS December 2022 expected
Advisor: Dr. R. Jacob Baker

Seoul Digital University, Computer Science BS, August 2016. graduated, 3.66 GPA

Work Experience

Software Engineer, Motional, Las Vegas, 2023 to present
Senior Controls Engineer, Imatrex Inc., Las Vegas, 2022 to present
- Developed mixed-circuit timing and control system for magnet coils and detector system.
- Researched and simulated image reconstruction algorithms and systems for various CT and digital tomosynthesis(DTS) concepts.
- C++/Python/GNU/Verilog

Staff Electrical and Software Engineer, Imatrex Inc., Las Vegas, 2021 to 2022
- Developed software for image & signal processing and data diagnostics.

Test Engineer, Imatrex Inc., Las Vegas, 2020 to 2021
- Tested & verified data acquisition systems and servers for high DTR communication between Xilinx FPGA devices and Linux servers.

Freelance Web Developer, 2016 to 2019
- Travel agency, wholesale clothing company, and etc.
- mySQL, HTML/Javascript/CSS/PHP

Personal Projects

Machine learning application to smart garage door opener for motorcycles
- Classifies specific motorcycles using CNN and opens the garage door with RF controller & IoT camera.
- Tensorflow/C++/image processing

Switching power supply integrated circuit design
- Flyback style delta-sigma method for reliable DC voltage generation.
- Cadence Virtuoso/LTSpice