Can Numba and CUDA Python Development Kits be used on a GPU to Speed Up 2D Gamma Index Calculations used in Radiotherapy?

Sun, Edward[1]

[1]Torrey Pines High School, San Diego, California January 29, 2023

1 Introduction

1.1 Radiotherapy Background

Cancer is the second leading cause of death following heart disease. In 2020 alone, cancer alone was responsible for the deaths of 602,350 deaths in the United states (**CDC**). However, with the aid of modern radiotherapy procedures, this figure can be significantly reduced. This procedure of radiotherapy is defined to include: patient consultation, simulation, treatment planning, and treatment delivery (**SBU**). In the patient consultation step, patients are initally diagnosed and informed of their condition. Following the consultation, the patients are taken to imaging facilities to capture images of their tumor(s) where contrast agents/chemicals are used to exaggerate the location of the tumor(s). Computer simulations are then used to define the location of the tumor and configurations of the treatment. Following this is the treatment planning step, where radiation oncologists, physicists, and dosimetrists are involved to plan the radiotherapy for the patient. The goal of this step is to deliver high doses to the tumor while limiting the damage to the surround health tissues, thereby reducing the side effects of radiotherapy. During this step, a commerical treatment planning system is often used to calculate treatment plans or how much radiation to send to a specific location (**Gardner**). Due to the sensitivity of radiotherapy, these treatment planning system is often rountinely requires the comparision of measured and calculated dose distributions (how the doses are delivered across a part of the patient's body). Given the tumor information of the patient, the treatment planning system will generate radiation delivery plans. However, to ensure the accuracy of dose delivery to the patient and minimize accidential radiation exposure, quality assurance (QA) is essential to treatment planning. Techniques involved in this quality assurance steps often involve the comparision of dose distributions of a measured dose distribution, obtained from running the treatment plan and irradiating a radiation detection matrix (fig XX), against the original treatment plan (**Low**).

2 Literature Review

2.1 QA Methods

Over the past decades, many metrics have been developed to improve QA. These methods are mathematical formulas/algorithms that often searches through and compares the calculated and measured dose distributions (dose images) in 2D or 3D to give a value that signifies the resemblance and therefore the accuracy of a treatment planner. For instance, Cheng *et.al*, used a pass-fail criteria for the dose difference (DD), which is the difference between the measured and calculated dose on a pixel, and distance to agreement (DTA), which is the distance/radius in which the algorithm searches on the dose distribution (**Cheng**). Each point within this search circle defined by the DTA is analyzed (Cheng). Similar metrics have been defined like the Global Gamma evaluation, maxiumum allowed dose difference (MADD) (Jiang), divide and conquer (D&C) (**Stojadinovic**), and more. However, one metric has been defined as the standard of QA metrics in the field of radiotherapy, that is the gamma index (**Low**).

2.2 γ index

The gamma index, originally proposed by Low et. al (**Low**), proposes a simple numeric method for. This formula is mathematically defined by the following:

$$\Gamma = \frac{|dose_{planned} - dose_{actual}|}{dose_{threshold}} \tag{1}$$

$$\tag{2}$$

This original formula for the simple gamma index is robust and reliable. The calculation is also relatively simple, as it is a simple subtraction, this calculation can be achieved with has been adopted and accepted widely. Since then, researchers have been making modifications to this algorithm adding more metrics for greater accuracy or changing the formula for greater efficency when running on computers' CPU or GPU (**Gu**).

One popular modification is to add DTA as a factor into the calculation of the index. Essentially, this modified algorithm is a iterative process which passes over every pixel on a dose distribution image. For the sake of easy understanding, we will use a 2D example, but the same logic can be applied for 3D dose distributions as well. To begin the search, we will start on one reference pixel (calculated dose distribution) and test pixel (measured dose distribution), where dose distributions in this case are defined as 2D grayscale images. Then, the iteration begins with defining a reference position in real units based on a pixel on the reference image and then iterating through the circle on the test image. Test pixel within this search radius is then evaluated with the formulas above along with a percentage passing criteria, which means that this pixel passes the γ index test if it is within this many percentage relative to the reference pixel. During this process, as the iteration uses real units (mm), it may generate a position that does not land on a pixel. Here, interpolation is used to estimate the value at that position within the search radius. This search radius for each reference pixel is actually the DTA and the percentage is the DTA criterion.

$$l = \sqrt{\frac{r^2(r_m, r)}{\Delta d_M^2} + \frac{\delta^2(r_m, r)}{\Delta D_M^2}} \tag{3}$$

$$r(r_m, r) = |r - r_m| \tag{4}$$

$$\delta(r_m, r) = D(r) - D_m(r_m) \tag{5}$$

$$\gamma(r_m) = min\{\Gamma(r_m, r_c)\forall(r_c)\} \tag{6}$$

Mathmatically, as seen in the equations above, Eq. (1) describes the intermediate value l calculated for each pixel, where $r^2(r_m, r)$ defines the square of the distance between the reference and the test pixel. The parameters r_m of $r^2(r_m, r)$ is the real position of the measured pixel and r defines the real position of the test pixel, as pointed out by Eq. (2). $\delta^2(r_m, r)$ defines the square of the difference in pixel value (in real terms, pixel value is radiation dose intensity) where r_m represents the pixel value of the measured pixel and r represents the pixel value of the test pixel, as pointed out by Eq. (3). $r^2(r_m, r)$ is then divded by Δd_M^2 which is the DTA or search radius. $\delta^2(r_m, r)$ is then divded by ΔD_M^2 which is the DTA criterion or acceptable passing percentage. This whole term is then square-rooted to produce the intermediate gamma value. This process is then repeated for all values within that radius for that refernce pixel. The final gamma index for that reference value is taken as the minimum value of all these previously calculated intermediate values. Then, this is repeated for all reference pixels from the reference image to produce one gamma value for each reference pixel, thereby creating a gamma image.

2.3 Challenges to the γ Index

With such a iterative and complex system involving intensive interpolation and calulations, Gu et. al (**Gu**) highlighted the concerns of speed with the γ index calculation, where clinically relevant dose distribution sizes is very time-consuming to compare. Many researchers have attempted to improve the accuracy and efficiency of the algorithm (**Bakai**, **Depuydt**, **Stock**, **Jiang**, **Spezi**) while others focused on speed (**Ju**, **Chen**, **Wendling**).

2.4 Code Acceleration in Python

Other than algorithmic approaches to speeding up the γ index, another approach is to use a graphics processing unit (GPU) to accelerate time consuming code. GPUs are a part of a modern computer that accelerates the rendering of computer graphics, especially in games, content creation, etc. (**intel**). However, they have also been repurposed for scientific computing, which takes advantage of their high calculation powers. As seen, GPUs have been used in weather prediction (**Michalakes**), mechanical structural analysis (**Georgescu**), and chemical analysis (**Ma**). These studies typicall use Nvidia CUDA acceleration toolkits in the C programming language or they are built from the ground up.

However, within recent years, Python has been a popular choice within the area of GPU acceleration. This is largely thanks to the growing development in acceleration libraries like Numba (**Lam**). By itself, Numba is a Just-in-Time (JIT) compiler for Python, where code (in this case python code) is compiled into machine code during the execution of the python program rather than before the execution of the program. This compilation into machine code makes python code much faster as Python by itself, is interpreted during run time on a virtual machine and then executed. In recent years, Nvidia, the manufacturer of a majority of commerical graphics cards, have partnered with the developers of Numbda to create GPU support for Numba, which now not only enables Numba to do JIT compilation, but also use GPUs to accelerate code.

Implementation of the gamma index in python isn't new. Many open-source repositories on github have produced efficient gamma index calculation scripts[1] Similarly, Biggs et. al (**Biggs**) have proposed and maintained a community driven project named pymedphys where CPU driven gamma index codes and other medical physics related codes have been implemented. GPU acceleration with gamma index isn't new either. For instance, Gu et. al had proposed gamma index acceleration with a GPU in 2005 (**Gu**). There have also been GPU acceleration toolkits released by MATLAB and researchers have implemented similar algorithms of the gamma index in MATLAB (**Matlab**). However, as seen, there is GPU acceleration in MATLAB, but not implemented with Numba and CUDA in python. Similarly, there have been gamma index implemented in python, however, there has not been GPU accelerated gamma index calculations.

[1]github.com/christopherpoole/pygamma

Python is one of the most popular data analysis and processing tool in the software engineering world. With the popularity of this language, more tools should be made avaliable for use in research. Thus, the reasoning behind the choice of this project is due to the high popularity of python, the large community support for GPU acceleration in python, and the prevalent use of the gamma index in radiotherapy. This study aims to fill this gap by creating a GPU accelerated gamma index framework for 1D, 2D, and 3D dose distributions in python via Numba and python CUDA. This will be achieved with a Nvidia GTX 1080 GPU (**NVidia**) running on PopOS 22.05 LTS with Nvidia CUDA support built in. For the algorithm, this study will be based off of Low et. al (**Low**)'s original gamma index paper witht no modifications made to the algorithm. This study aims to test the feasibility of implementing the gamma index with Numba and CUDA while also benchmarking this against other frameworks within other languages (MATLAB, C, etc) to analyze if Python Numba and CUDA accelerated gamma index will be competitive compared to these other languages' frameworks. As this study will be benchmarking and testing a framework with a limited set of testing data, it will be suitable for an experimental design methodology. In this next section, the specific methodologies will be more clearly defined.

3 Methods

3.1 Choice of Language

In traditional GPU-acceleration of the γ index (**Gu**) programming languages like C/C++ or MATLAB are often used. C/C++ similarly has CUDA development support from NVidia, but the syntax is often more complicated than that used by Python. MATLAB has its own GPU Acceleration toolkit as well, but MATLAB is a expensive paid service which may make it unfeasible for some researchers who may not have the resources to purchase such a service. Thus, Python is a better language of choice for this task due to is easy syntax to implement, which may prove beneficial for researchers that want to understand or improve the source code of this project, and it is free and open-source for anyone to download and use. But the main downfall of Python as oppose to C/C++ or MATLAB is its slow speed due to Python being an interpreted language as explained earlier. Thus, it is important to also investigate weather or not the GPU acceleration through Numba and

CUDA will be significant enough to overcome these performance differences based on their inherit language designs.

3.2 Program Design

As there are many modified algorithms out in the research community on how to calculate the γ index, I chose to write accelerated code for the version based on only the dose differences, dose threshold, and DTA (Eqn. 1 and 2) and the modified version of **Low**. (Eqn. 3 -6). I decided to select the first version of the calculation based of its simplicity and popularity and the second I selected based off its relative accuracy as well as its popularity as many other researchers have used these formulas in their works (**Gu**).

First, for the simpler only dose difference based calculation, the operation of finding the dose difference was achieved through the following simple matrix subtraction, where each matrix is the input 2D dose distribution. Then, the DTA criteria is implemented via the following formula:

$$(TBD, Debuggingcodern) \tag{7}$$
$$\tag{8}$$

Then, this is combined with the regular gamma index formula through only expressing the minimum value between this calulated DTA and the gamma index.

3.3 Benchmarking

For comparing the feasbility of this accelerated gamma index calculation, I compared multiple open-source projects all attempting to achieve the same purpose of calculating the γ index. Specifically, I used projects written in Python and MATLAB primarily to compare to my codes. For Python, I used pygamma by christopherpoole [1], gamma-index by janpipek [2], gamma-py also

[1]github.com/christopherpoole/pygamma

[2]github.com/janpipek/gamma_index

by janpipek [3], pymedphys [5] and also my own versions of non-accelerated loop-based γ index calculation (as detailed in the γ index section of this paper) and a matrix version the calculation. For MATLAB, I used mwgeurts' CalcGamma [5]. In total, there are 7 comparision codes that I ran and timed as shown in Table 1. These are used as a benchmark to later compare with the accelerated of the calculation.

[3]gist.github.com/janpipek/334c2533b87cd75c3f59

[5]https://github.com/pymedphys/pymedphys

[5]https://github.com/mwgeurts/gamma