Using Futhark for a fast, parallel implementation of the Simultaneous Iterative Reconstruction Technique - A pre-study

Mette Bjerg and Lærke Pedersen

October 28, 2018

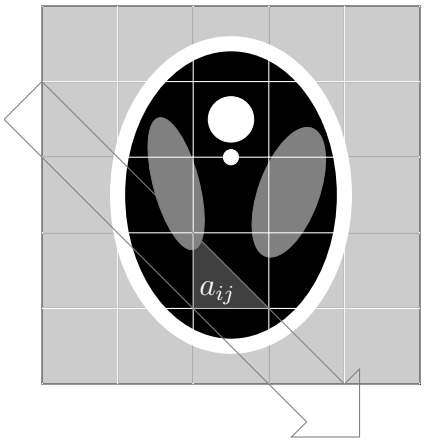

Figure 1: Example of a weighting a_{ij}.

1 Introduction

Computed tomography is the inverse problem of reconstructing an image or volume from its x-ray projections. The x-ray source spins around the object to be analyzed and sends x-rays that hit a detector on the oposite side. The detector shows how much of each x-ray was attenuated when passing through the volume. In this report we will focus on a type of algebraic reconstruction algorithm called the Simulataneous Iterative Reconstruction Algorithm (SIRT) and investigate how we may use the high level data-parallel language Futhark for implememting a fast version of this algorithm. The reason for chosing this algorithm is that it provides good reconstruction quality under non-optimal circumstances, but unfortunately has very poor performance and therefore is rarely used in practice .

give a citation for benefits and bad stuff of SIRT

In algebraic reconstruction we solve the problem as a linear system of equations. The main idea is that the process can be modelled as a linear transformation by discretizng the object to be reconstructed into N pixels.First we place a coordinate system with origo at the center of the object to be reconstructed and denote by $\boldsymbol{\theta}$ the vector of angles between the positions of the source and origo. For each angles several x-rays are cast from the source. We denote by the vector $\boldsymbol{\rho}$ the signed distances from each line to origo. The data produced by the process is called the sinogram. Then the sinogram values p_i for each (θ_k, ρ_l) are a weighted sum of the attenuations at each pixel f_j that the (θ_k, ρ_l) ray passes through:

$$\sum_{j=1}^{N} a_{ij} f_j = p_i \tag{1}$$

Where a_{ij} are the weights, corresponding to the fraction of the pixel j that the ray i covers.

Writing all the projections as a column vector $\boldsymbol{p}$ and the attenuation values to be reconstructed as a column vector $\boldsymbol{f}$ the weightings are represented as an $M \times N$ matrix

$\boldsymbol{A}$, we obtain a linear system of the form:

$$\boldsymbol{p} = \boldsymbol{A}\boldsymbol{f} \tag{2}$$

These systems of equations may easily be solved under the right circumstances, where $M = N$. However this is rarely the case. In most real cases $M > N$, i.e. the number of projections is larger than the number of pixels to be reconstructed and the size of the matrix is very large - more about this in the next section.

However, the algebraic reconstruction methods also have some advantages. Since the model closely relates to the real world scenario the weightings can be refactored to take irregularities in the setup, such as differences in beam energies or irregular geometries and missing data into account. Furthermore these methods generally give better image quality than analytic methods when the data is sparse.

A system like this is typically solved by minimizing some norm:

$$\|\boldsymbol{A}\boldsymbol{f} - \boldsymbol{p}\| \tag{3}$$

An example is the SIRT algorithm. The action of the matrix $\boldsymbol{A}$ is called the *forward projection*, and the matrix itself is called the *system matrix*. Each row of $\boldsymbol{A}$ represents the coefficients of the equation for one ray. The transpose $\boldsymbol{A}^T$ is called the backprojections, and can be vizualized as smearing the projection values across the reconstruction. The idea behind the SIRT algorithm is to forward project the current reconstruction, then subtract this from the original projection data and do a weighted backprojection resulting in a correction factor which can be added to the current reconstruction. The update equeation is:

$$\boldsymbol{f}^n = \boldsymbol{f}^{(n-1)} + \boldsymbol{C}\boldsymbol{A}^T\boldsymbol{R}(\boldsymbol{p} - \boldsymbol{A}\boldsymbol{f}^{(n-1)}), \tag{4}$$

where $\boldsymbol{C}$ and $\boldsymbol{R}$ are the diagonal matrices containing the inverse column and row sums of the system matrix respectively.

It can be shown that this iterative scheme solves the problem:

$$\boldsymbol{f}^* = argmin_{\boldsymbol{f}} \|\boldsymbol{p} - \boldsymbol{A}\boldsymbol{f}\|_{\boldsymbol{R}}, \tag{5}$$

where $\|\boldsymbol{x}\|_{\boldsymbol{R}} = \boldsymbol{x}^T\boldsymbol{R}\boldsymbol{x}$.

The backprojection and foward projection operations are standard operations in many iterative algebraic reconstruction methods and are the bottle necks of the algorithms . Therefore, our main focus has been on optimizing these operations, and then combining

insert citation here or just do runtime analysis of the different parts!

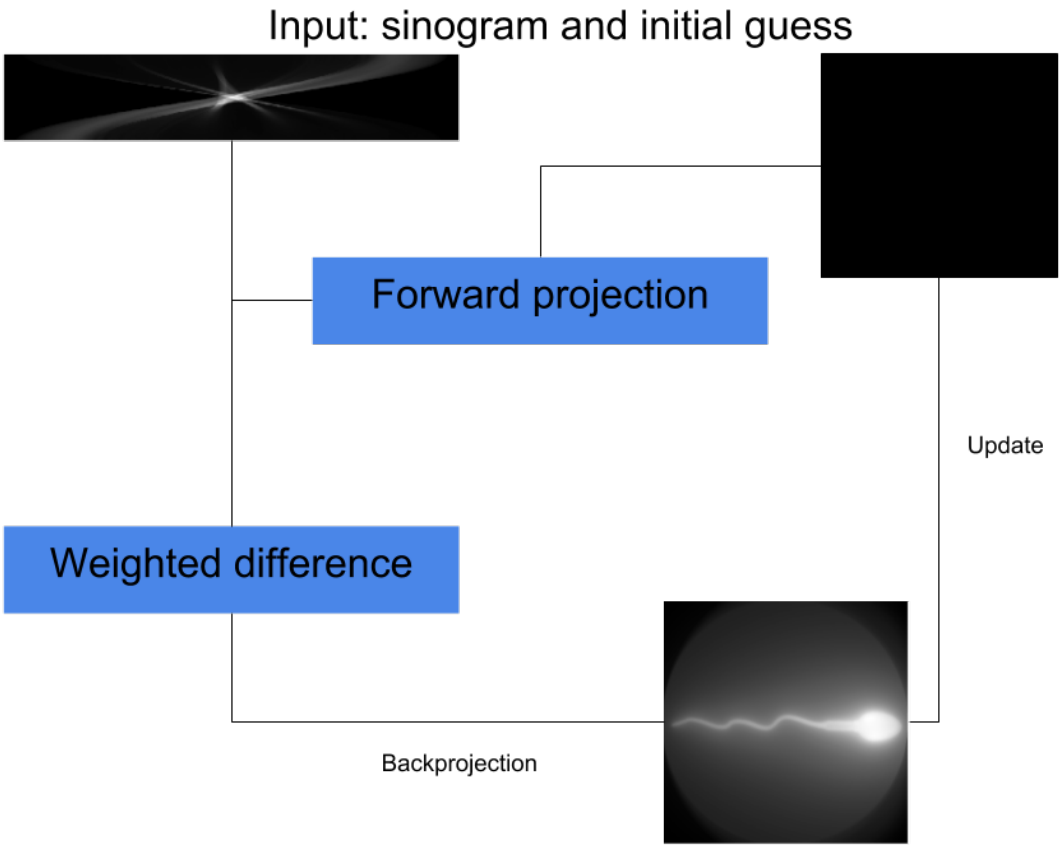

Figure 2: An illustration of the SIRT algorithm.

these to the SIRT algorithm.
Different beam geometries exist, such as parallel beams, fan beams and cone beams. The geometry must be considered when constructing the system matrix. However as this study is intended for applications with synchrotron data only parallel beam geometries will be considered. Furthermore, we will only consider reconstructions of 2D imaged.

2 Problem size

One of the main problems when parallelizing the algorithm is that the amount of data in real applications is huge. Images from synchrotrons are generated with detectors of sizes up to 4000×4000. To accurately generate 3D reconstructions it has been proven that approximately $\frac{\pi \cdot N}{2}$ vieweing angles are needed, where N is the size of the detector in one direction . To reconstruct an object
We will benchmark the algorithms by using sizes N ranging from 128 to 4096 and use $\frac{\pi \cdot N}{2}$ angles and N lines for each of these sizes.
To solve the large problems it is not possible to store the whole system matrix on the GPU, therefore part of solving the problem also involves computing the system matrix as we go along. For this we used the code from a bachelor project. . We had two different implementations which we compared, and we decided to use
A looped version of the forward projection with system matrix cut in *steps* chunks looks like this:

```
1 for step = 0; step < steps; step++
2         A = getRays(raysperstep)
3         for ray = 0; ray < raysperstep; row++
4                 acc = 0.0
5                 for p = 0; p<numpixels; p++
6                         acc+= A[ray][p]*image[p]
7                 FP[step*raysperstep+ray] = acc
```

Figure 3: A looped version of the forward projection, where the rowsperstep should be the largest number possible such that the computations fit in the memory. $step * rowsperstep$ should equal the total number of rows.

A looped version of the backprojections looks like this: A looped version of the forward projection with system matrix cut in chunks looks like this:

3 Flattening

Exploiting nested parallelism, as in the code sketched in the previous section, is difficult on GPU since the hardware is organized on one or two parallel levels that allow threads to comminucate via shared scratchpad memory.

insert citation for the number of angles needed
ceil of angles
write about how much memory is on normal GPU and how much the matrix would use
elaborate on this - how did they solve it
fill out which one we used and why.
Maybe with a graph showing the performance of the two, vs. francois version.
and some

```
1  for step = 0; step < steps; step++
2    A = getRays(raysperstep)
3    AT = A.transpose()
4    for p = 0; p<numpixels; p++
5         acc = 0.0
6         for ray = 0; ray<raysperstep; ray++
7                                  acc+= AT[p][ray]*sinogram[ray]
8         BP[p] += acc
```

Figure 4: A looped version of the forward projection, where the rowsperstep should be the largest number possible such that the computations fit in the memory. $step * rowsperstep$ should equal the total number of rows.

One way to get around this problem is to use a flattening transformation. The problem with this is that it will require even more memory usage, and may prevent opportunities for locality optimizations .

relate to our problem

4 Comparison to a CUDA implementation

We compared our implementation of forward projection and back projection to implementations from the astra toolbox.

make graphs displaying run times and memory usage of flattened versions vs. looped versions. Insert pseudocode of flattened versions cite article about astra and write about how these are implemented - look into code/read articles from mat mat mult assignment it is clear that memory

5 Memory coalescence and other optimization ideas

References

[1] Z. Xue, L. Zhang, and J. Pan. A new algorithm for calculating the radiological path in ct image reconstruction. In *Proceedings of 2011 International Conference on Electronic Mechanical Engineering and Information Technology*, volume 9, pages 4527–4530, Aug 2011.