X-ray CT Reconstruction using the Julia Language

Samuel Rohrer (Honors Capstone) & Jeffrey Fessler

Electrical Engineering and Computer Science Department, University of Michigan

Problem

Deploying computationally high performance scientific computing code traditionally requires two distinct steps:

1. Prototype new algorithms in a high level language, such as Python or MATLAB, to verify functionality.
2. Translate to a lower level language such as C or C++ to empirically determine performance gains.

Reconstruction of X-ray computed tomography (CT) scans is an example of high performance scientific computing code that requires both steps.

Goal

The **Julia** numerical computing language aims to combine these coding steps [1]. The **Julia** language can be applied to medical imaging reconstruction (inverse) problems by allowing native calls to already written C library code that optionally uses GPUs for acceleration.

This project aims to write key components of X-ray CT reconstruction algorithms in **Julia**, based on existing MATLAB code.

Introduction

Iterative X-ray CT reconstruction algorithms can improve image quality for low dose scans. However, their clinical utility has been hampered by their enormous computational requirements; typical low-dose reconstructions require about an hour on commercial systems. Faster reconstruction times will enable ubiquitous use of low-dose CT. [2]

The **Michigan Image Reconstruction Toolbox** was created to evaluate reconstruction algorithms, including X-ray CT [3]. Algorithms can be run purely in MATLAB, or accelerated by calling compiled C code. However this requires *glue code*, which can cause unnecessary computational overhead.

Julia uses a just-in-time compilation backend that allows substitution of compiled C code machine instructions while creating compiled binaries. This means there is no *glue code*.

CT Problem Setup

The inputs to the X-ray CT reconstruction problem are the CT system geometry and X-ray projections through the *image volume*, shown below:

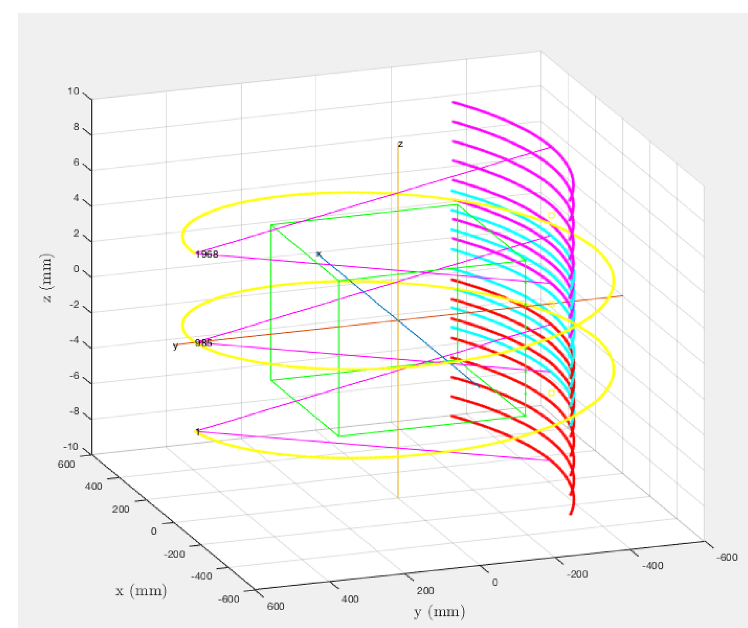

Figure 1: Typical CT scanner setup. *Image volume* (patient) is green box.

To iteratively reconstruct an approximate image using model based reconstruction two major operations are required:

1. **Forward** projection of approximate image through *image volume* to determine approximate projections.
2. **Backward** projection of approximate projections back through *image volume* to recover an approximate original image.

These operations continue until the forward projection of the approximated image matches the measured X-ray projections.

C Library Call Results

The time to execute required operations for iterative reconstruction using compiled C code is shown below:

System Time (seconds)		
Operation	**MATLAB**	**Julia**
Memory Alloc	0.067	0.483
Forward Proj	26.096	24.235
Backward Proj	23.614	20.159

Julia `ccall`

Julia provides a native interface to call compiled C code, available through the `ccall` command. An example to allocate memory follows:

```
ret = ccall((:malloc, stdlib),
            Cint, (Csize_t), mem_ptr )
```

This `ccall` allocates memory by using the C function `malloc` with memory of size `mem_ptr` and returns the new memory pointer in the variable `ret`. To call an arbitrary C function the C code must be compiled as a shared library (default for most software distributions), then a similar `ccall` in **Julia** can be used.

Nesterov Accelerated Weighted Least Squares

Determining the approximate image that created the measured X-ray projections can be written as the solution to the weighted least squares minimization problem. W is the weight matrix, A is the system matrix, y_{proj} is the measured X-ray projections, and x_{im} is the approximate image.

$$\hat{X}_{true} = \text{argmin } \frac{1}{2}||y_{proj} - Ax_{im}||_W^2$$

Differentiating this cost function yields the gradient:

$$\nabla f = A^T W(Ax_{im} - y_{proj})$$

Finally, applying Nesterov's fast gradient method yields the following update step:

$$t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2}$$

$$z_{k+1} = x_k + (\frac{t_k - 1}{t_{k+1}})(x_k - x_{k-1})$$

$$x_{k+1} = z_{k+1} - \alpha \nabla f$$

This algorithm is easily implemented and correctly reconstructs a CT image.

Conclusion

Using the **Julia** language it is possible to natively call compiled C libraries to reconstruct X-ray CT images. Native C calls from **Julia** are marginally faster than the equivalent MATLAB calls. However, the **Julia** calls require no additional developer time to implement. MATLAB requires the development of *glue code* and additional steps to build the C libraries correctly.

Future Work

1. Remove unnecessary memory allocations in **Julia** from C library call function wrappers.
2. Extend `ccall` function wrappers to use GPU based algorithms.
3. Compare results to algorithms written natively in **Julia**.

Acknowledgements

Would like to thank:

- Professor Jeffrey Fessler for his mentorship and teaching.
- Engineering Honors office for their support of the project, and Rachel Armstrong's support in finding this project.
- EECS Department.

All code used in this research will become open-source.

References

[1] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah.
Julia: A fresh approach to numerical computing.
SIAM Review, 59(1):65–98, 2017.

[2] Jeffrey M Rosen, Junjie Wu, TF Wenisch, and JA Fessler.
Iterative helical ct reconstruction in the cloud for ten dollars in five minutes.
In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, pages 241–4, 2013.

[3] Jeffrey Fessler et al.
Michigan image reconstruction toolbox, 2017.