Accelerating Computed Tomography using OpenMP and CUDA

Team:
1. HEMANTH SRIDHAR NAKSHATRI (nakshatri@wisc.edu)
2. HARSHIL OZA (hboza@wisc.edu)

12/02/2024

Introduction: What and Why

- This project turbocharges parallel beam CT image reconstruction using OpenMP for parallel CPU processing and CUDA for GPU acceleration.

- We address the inefficiencies in Radon transform and backprojection algorithms, ensuring scalable and high-performance processing for complex imaging tasks.

- Faster CT imaging enables real-time diagnostics, better resource utilization, and improved outcomes, impacting healthcare and industries that rely on precise imaging technologies.

Challenges

- **Performance Bottlenecks:**
  - Radon transform and Backprojection algorithms are computationally expensive.
    - Atleast O(n^2 log(n)) in time complexity.
  - Other operations like FFT, Bilinear interpolations are not so easy on the compute resources when considering the whole pipeline.

- **Scalability:**
  - Naive single-threaded implementation does not scale well with increase in complexity of the system.

- **BMP Files:**
  - In an attempt to move away from OpenCV dependency issues, we decided to implement methods to read and write *.bmp* files without needing any additional libraries.

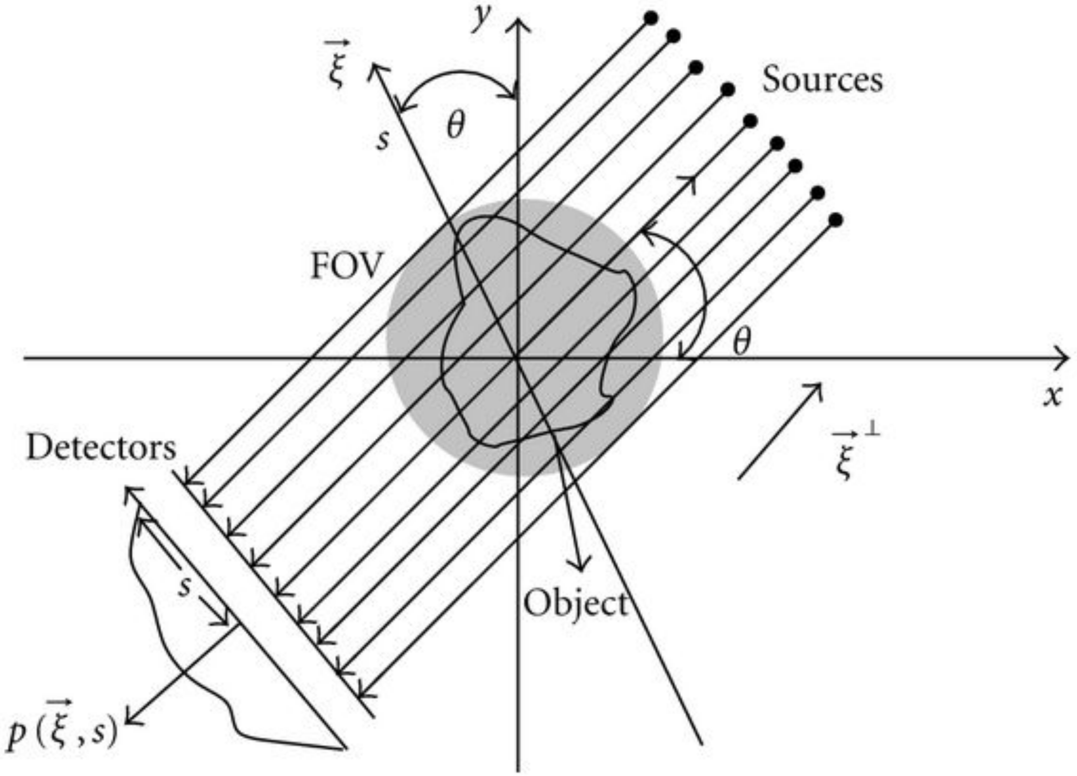
Implementation Details

- **Computed Tomography (CT)** mainly has two parts mathematically.
  - **Forward modeling:** Radon Transform
    - Converts a 2D image of an object into a set of projections taken at different angles. Creates a "Line Integral" of the object along many different lines.
    - More projections from many viewing angles gives better results. But adds significant compute time.
  - **Backprojection:** Inverse Radon Transform
    - Converts the projected data back to image domain.
- Optimized the code and loops for it to have both spatial and temporal locality.
  - Improved the overall results across all configurations.
- *Optional, but necessary intermediate step:*
  - **Ramp Filtering:** Necessary to eliminate hazy and blurred reconstructed results.
    - Accentuates sharp edges and details by amplifying high frequency components and suppressing low frequency components.

- All of these were further parallelized using OpenMP and CUDA as well.
  - Tried basic optimizations like scheduling, collapse, simd etc.

For more details:
https://towardsdatascience.com/the-radon-transform-basic-principle-3179b33f773a

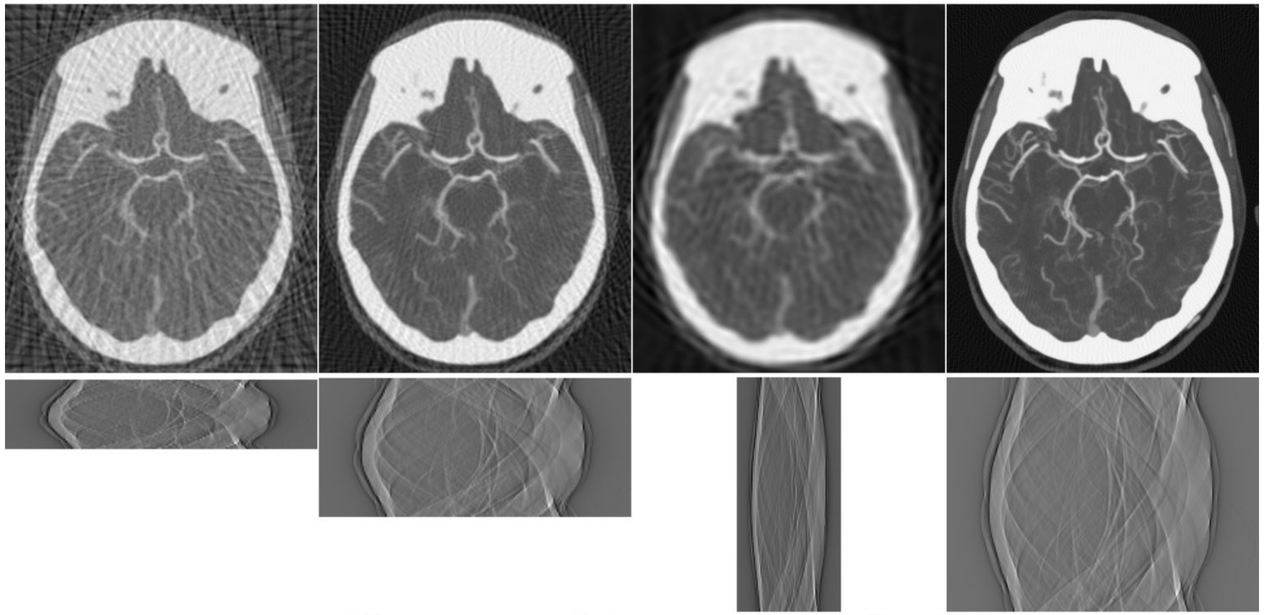
Experimental Results: Images



Parallel Beam CT

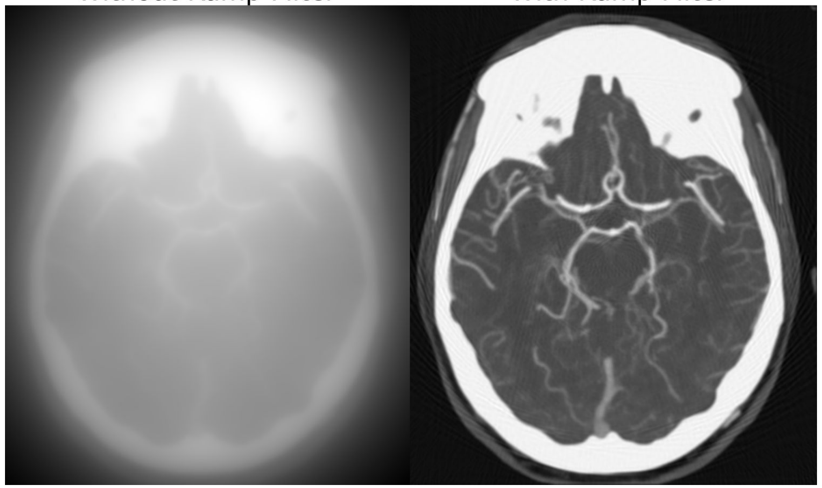
https://bigwww.epfl.ch/teaching/projects/abstract.html?f=00359

Reconstucted Images (Filtered Backprojection)

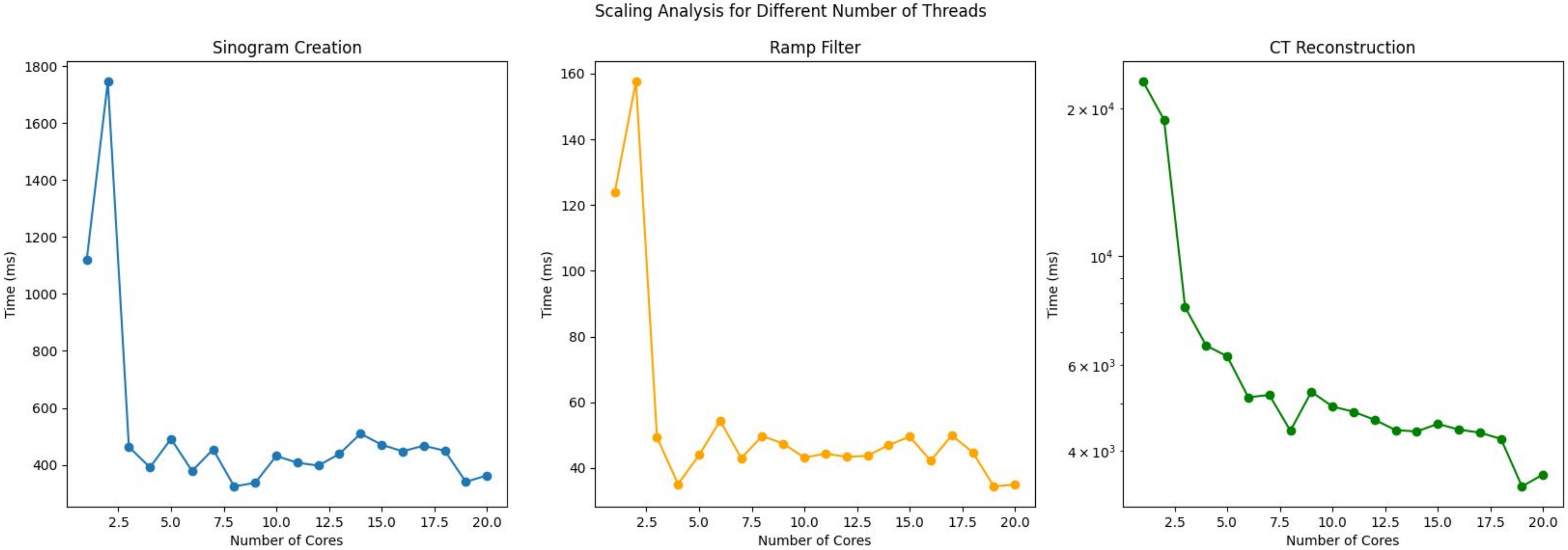


Sinograms (Forward Modeling)

Without Ramp Filter　　With Ramp Filter

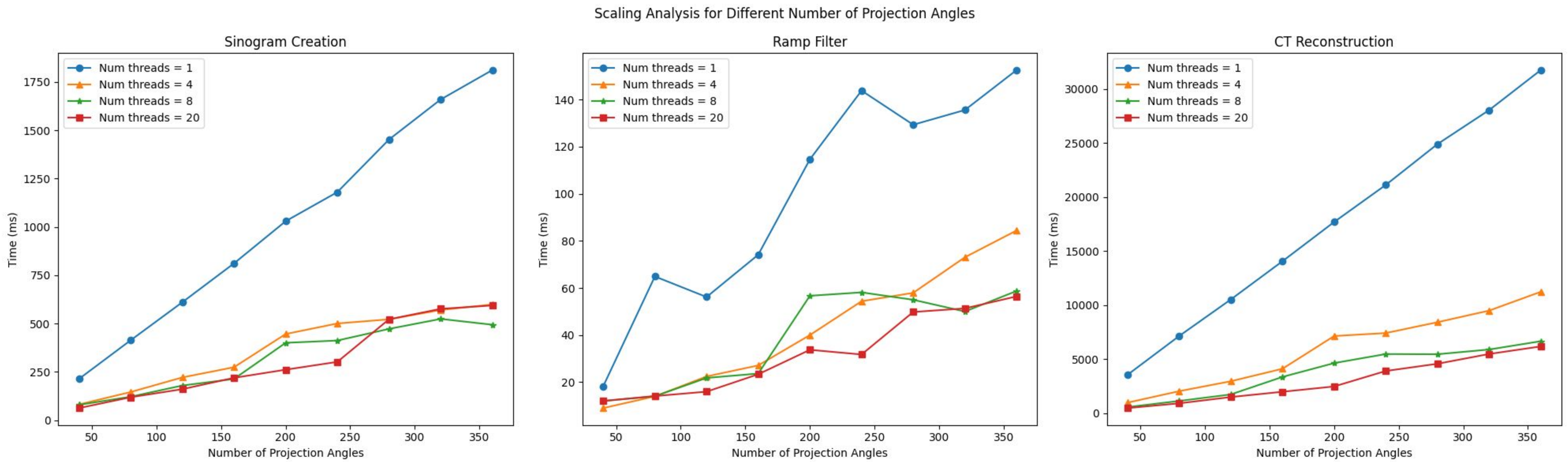


Experimental Results: OpenMP



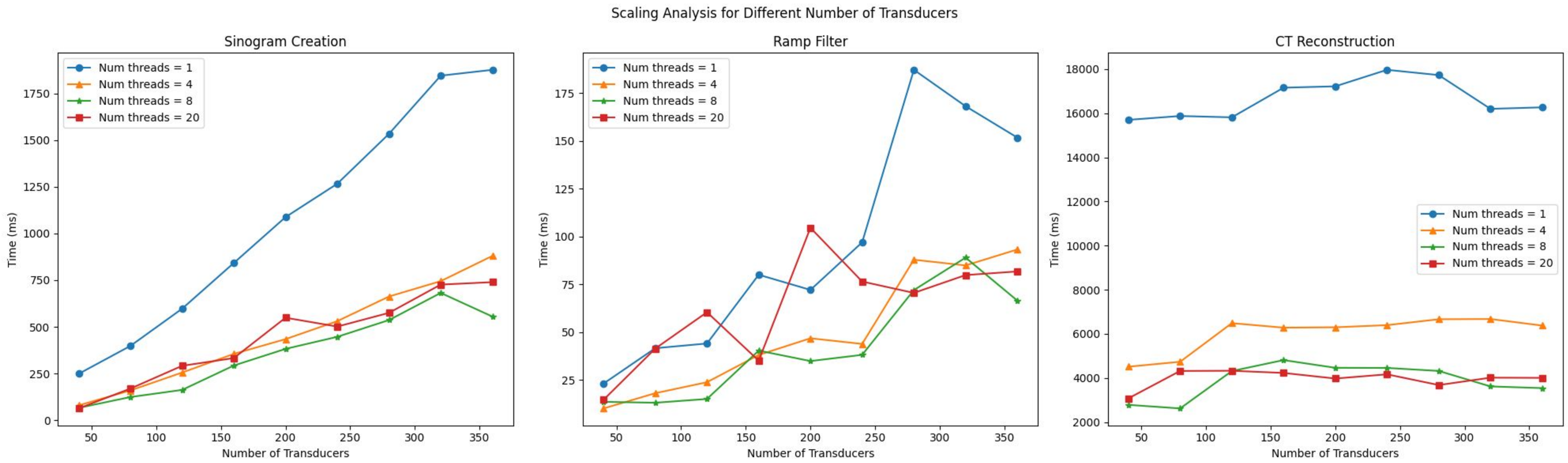
- **OpenMP Implementation**
  - Compute time for the main stages with increase number of threads.
    - We note a general decrease in compute times for all stages
    - Diminishing return with more threads.

Experimental Results: OpenMP



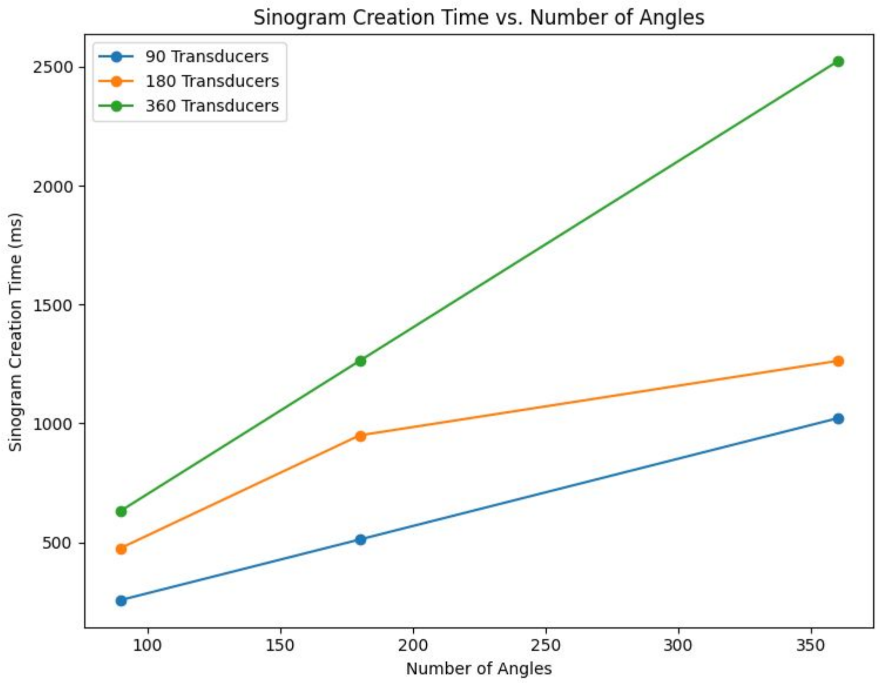
- **OpenMP Implementation**
  - Compute time analysis with increased complexity (Increasing number of viewing angles)
    - Increase in compute time with increase in transducers.
    - But we see a trend of decrease in time with more threads.

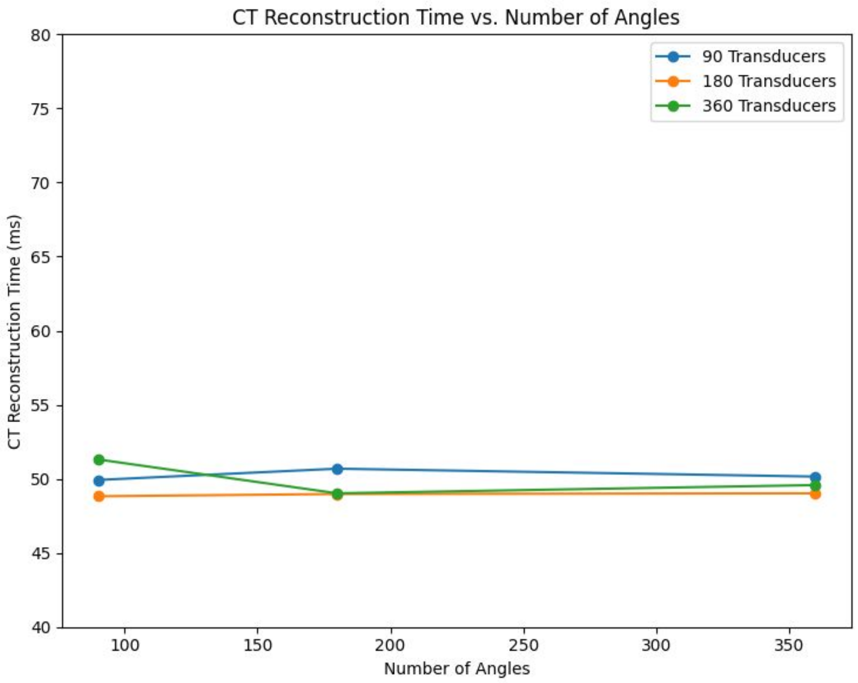
Experimental Results: OpenMP



- **OpenMP Implementation**
  - Compute time analysis with increased complexity (Increasing number of transducers)
    - Increase in compute time with increase in transducers.
    - But we see a trend of decrease in time with more threads.
- ***NOTE:*** *The last image has negligible increase in time due to memory access pattern of our algorithm.*

Experimental Results: CUDA





- **CUDA Implementation**
  - Similar to OpenMP implementation for Forward modeling (Sinogram Creation).
  - Extremely efficient in Backprojection (Reconstruction) for almost all configurations.
    - Almost 1000x faster than CPU implementations.

Takeaway

- **Technical Implementations:**
  - Hybrid CPU-GPU model makes the solution optimal.
    - Handle simpler tasks with CPU and computationally expensive tasks with GPU.
    - Reduces unnecessary overheads and data transfers between devices.

- **Collaboration pays off:**
  - Dividing tasks into modular components made our debugging efficient.
  - Team brainstorming gave us creative ideas to tackle algorithmic issues.

- Learned to design scalable systems with portable custom solutions, eliminating external dependencies for better adaptability across platforms.

THANK YOU!