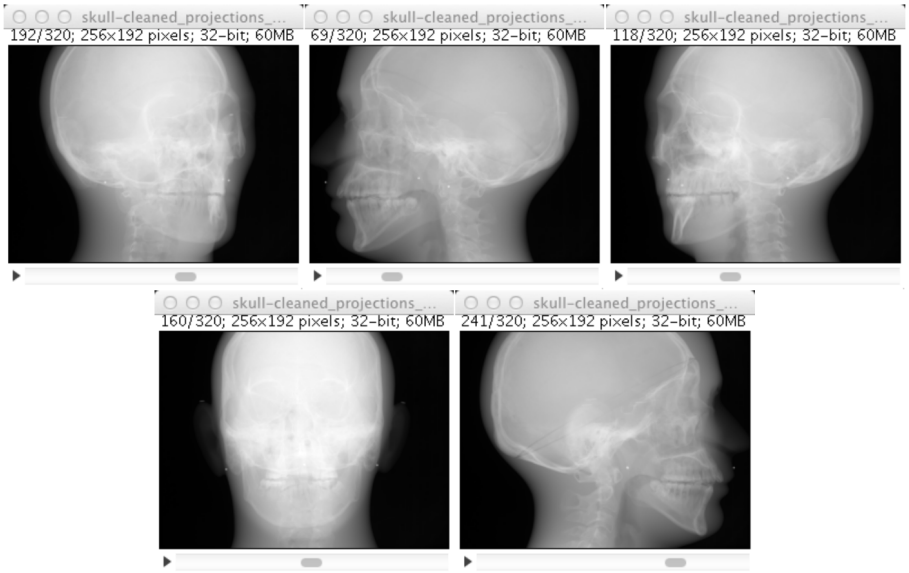
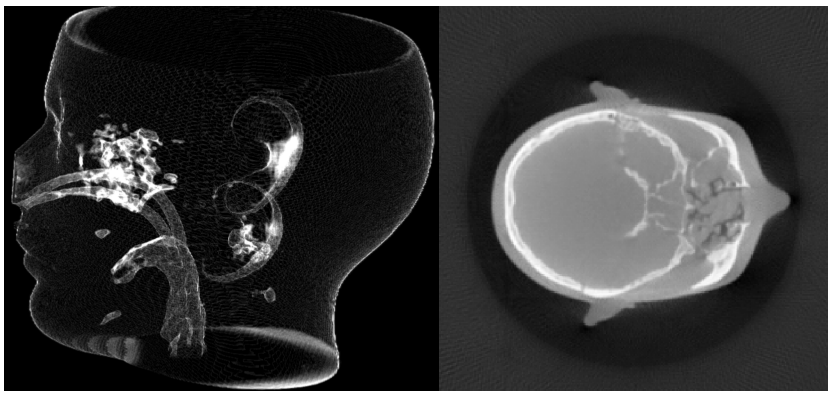
Assignment 4: CT Reconstruction

Copenhagen, March the 24th, 2021



2D CT Scans



A slice of a 3D CT reconstruction

1 Practical Information

- Deadline: 11/4 23:59.
- Resources:

  ERDA for file storage[1].

  Jupiter for the *Terminal* to access DAG and MODI.

  DAG for testing, visualization, and debugging.

  MODI for producing performance numbers for your report.

2 Introduction

CT-Reconstruction is an important quantitating and visualising tool for all natural sciences. To learn the basics we use a cone-beam step-and-shoot method based on the inverse-radon transform that is a mathematical model for transforming a series of 2D images into a 3D volume.

2.1 CT Data Files

Reconstructing a 3D volume from a series of 2D images requires several steps, in this assignment we will only look at the core steps. In order to simplify the task a number of pre-calculated input files are provided with the assignment:

```
projections.bin
combined.bin
z_voxel_coords.bin
transform.bin
volumeweight.bin
```

`projections.bin` Matrix with the pre-processed 2D X-Ray images.

`combined.bin` Matrix with all combinations of all X,Y coordinates for the 3D volume.

`z_voxel_coords.bin` Matrix with Z coordinates for the 3D volume.

`transform.bin` Matrix used to map 3D coordinates to 2D coordinates.

`volumeweight.bin` Post weight to compensate for the cone effect of the X-Ray beam.

The files a located `~/modi_readonly/ct_data` on MODI. In order to debug on DAG, we have also included some smaller data files at `~/hpc_course/module5/CT_Reconstruction/ct_data`.

[1] `http://www.erda.dk/`

2.2 Help Files

In ERDA/Jupyter you have access to some help files – in the path `~/hpc_course/module5/CT_Reconstruction` you will find the following:

```
Makefile
ct_parallel.cpp
ct_sequential.cpp
visual.py
slurm_job.sh
ct_data
```

`Makefile` is for compiling the programs.

`ct_sequential.cpp` is the fully implemented sequential version. Notice, even though this is a sequential version, it still uses MPI to read and write files.

`ct_parallel.cpp` is the start of the parallel version that you should finish.

`visual.py` is a Python program for visualizing the MPI written file generated by `ct_sequential.cpp` and `ct_parallel.cpp`.

`slurm_job.sh` is an example of a SLURM script that runs with 256 voxels cubed.

`ct_data` small data set for testing and debugging on DAG. On MODI use `~/modi_readonly/ct_data`.

In order to compile and run the sequential version, after copying files to your one working directory, you can do:

```
cd ~/erda_mount/hpc_course/module5/CT_Reconstruction

# Compile the two versions
make

# Run the sequential version with a 16 voxel cube.
# and writing the output to data.bin
./ct_parallel --num-voxels 16 --input ct_data --out data.bin

# Convert data.bin into a gif video
python visual.py data.bin data.gif
```

NB: Remember to move your own source files into `~/erda_mount`. Every modified file in the *root* will be reset on restart. Hint, move the directory into ERDA and work from there:

```
cp -a ~/hpc_course/module5/CT_Reconstruction ~/erda_mount/hpc_course/
```

2.3 Correctness

In order to check the correctness of your solution, you can use the checksum printed when setting the `--out` argument. This checksum should be the same (or very close) for the sequential and the parallel version when running with the same arguments.

3 The Assignment

The assignment is to implement a parallel version of the sequential CT reconstruction using MPI and OpenMP. The implementation should use MPI to parallelise over projections and OpenMP to parallelise each projection calculation.

In order to handle all MPI-processes reading the projection file concurrently, you should use MPI-IO to implement the file read. Notice, we do not use HDF5 in this assignment – a corresponding HDF5 algorithm would require the use of HDF5 hyperslabs.

Beside your implementation, you must submit a report through Absalon. The report should be no longer than 3 pages in total. You should explain how you have parallelised the program and how this has influenced your performance. You may time the OpenMP and the MPI-IO part individually.

You should provide experiment results of running using various number of MPI-processes and OpenMP threads. The exact runs are up to you, but remember to discuss your decisions and the results.

The results should be presented as an easy to read graph, which includes the absolute and relative before and after transformation of the code. **NB:** when benchmarking performance, do not write any output file!

4 Where to Run?

Test & Debug

When testing, debugging, and visualising your program, you should use the **DAG** servers through Jupyter. Remember to use the small CT data set: `~/erda_mount/hpc_course/module5/CT_Reconstruction/ct_data`.

Benchmark

When benchmarking the performance of you program, use the **MODI** servers also through Jupyter. However, in order to get exclusive access to a machine you need to submit your run through SLURM.

The MODI cluster:

- Eight nodes and a frontend.
- AMD EPYC 7501 @ 2Ghz.

- 2×32 CPU-cores.
- 256GB of Memory.
- 25Gbit Ethernet RoCE.
  - *RDMA over Converged Ethernet.*
  - Infiniband like.
  - RDMA at $1.2\mu s$.

In order to run on the MODI nodes, you need to copy `ct_parallel` to the MODI mount: `~/modi_mount`. An example of a SLURM script, `slurm_job.sh`, that runs with 256 voxels:

```
#!/bin/bash
#SBATCH --exclusive

# set loop scheduling to static
export OMP_SCHEDULE=static

# Schedule one thread per core. Change to "threads" for hyperthreading
export OMP_PLACES=cores

# Place threads as close to each other as possible
export OMP_PROC_BIND=close

# Set number of OpenMP thread to use (this should be 64 cores / number of ranks pr node)
export OMP_NUM_THREADS=32

# Run the program
mpirun singularity exec ~/modi_images/hpc-notebook_latest.sif ./ct_parallel \
    --num-voxels 256 --input ~/modi_readonly/ct_data
```

In order to run the simulation using four MPI-processes `-n 4` on two MODI nodes `-N 2` (two MPI-processes per node), run the following command:

```
sbatch -n 4 -N 2 slurm_job.sh
```

When finished, the output is written to `~/modi_mount/slurm-<ID>.out`, where `<ID>` is the ID given by SLURM when submitting the job.

```
CT Reconstruction running on `modi000.science`, rank 0 out of 4.
CT Reconstruction running on `modi000.science`, rank 1 out of 4.
CT Reconstruction running on `modi001.science`, rank 2 out of 4.
CT Reconstruction running on `modi001.science`, rank 3 out of 4.
elapsed time: 4.66597 sec
```