Bickston and Yanni

Reproducibility Lab
VIP - Team Phoenix
Spring 2023
Due Date: April 19th

Resources:

- MemXCT-CPU source code: https://github.com/merthidayetoglu/MemXCT-CPU
- MemXCT-GPU source code: https://github.com/merthidayetoglu/MemXCT-GPU
- ICE Cluster hardware specifications: https://docs.pace.gatech.edu/ice_cluster/ice/
- ICE Cluster student user guide: https://docs.pace.gatech.edu/ice_cluster/ice-guide/

Learning Objectives:

Reproducibility is an important aspect to the scientific community. It is important to make sure that, under the same experimental conditions, that experiment results can be reproduced and validated to be correct. Thus, reproducibility is one of the challenges of supercomputing competitions. The purpose of this lab is threefold:

- Teach you how to read a scientific paper to understand it’s key contributions and experimental results
- Experiment with building scientific applications from source code using different libraries, compiler chains, and hardware
- Conduct computational experiments and report your results in a clear, concise way

Steps: (Use coc-ice for all experiments)

1. Read the MemXCT paper using this link (https://drive.google.com/file/d/1HQB9jXcPT8Q2zXkZn9ychSl6NdTb6ydu/view?usp=sharing), and answer the following questions
    a. What is XCT and how is it used in the real world?
    b. What is the difference between a tomogram and a sinogram?
    c. What are the major bottlenecks of conventional XCT approaches, and how does this paper approach the problem differently?
    d. Explain two of the optimizations that MemXCT uses to improve performance.
2. Download the MemXCT-CPU and MemXCT-GPU repositories (listed above), as well as all the ADS* and CDS* datasets.
    a. Note: you may want to test using the ADS1 and ADS2 datasets
3. Each of the repositories should have a couple makefiles in them. Copy one of the makefiles to use on coc-ice, and modify it to be able to correctly build on coc-ice hardware.
    a. Hint: Make sure your makefile includes the correct compiler based on what you `module load`.
    b. Hint: For MemXCT-GPU, make sure that the nvcc “arch” flag matches the architecture of the GPU you decide to run on

Experiment 1: Tuning

- Using any one of the datasets of your choice, tune the single node CPU performance of MemXCT. Experiment with different compilers and mpi libraries. Also experiment with different Tile Sizes, Block Sizes, and Buffer Sizes. Report your results.
    - Hint: copy one of the `run.sh` files and modify it to run on coc-ice
    - Hint: for every experimental setup, run the trial 3 times and take the average.
    - Hint: there are 5 degrees of freedom here (tile size, block size, buffer size, C/C++ compiler, and mpi library), meaning there should be at least 2^5 different experimental setups. However, it is probably better to try more than just 2 tile size, block size, and buffer sizes. Keep this in mind when thinking about how to automate your experiments and data collection, and also keep that in mind when deciding which dataset to use
- In your report, explain what each of the different runtime parameters (tile, block, and buffer size) does, and how changing these values affects performance.

Experiment 2: CPU Scaling

- Using the CDS2 and CDS3 datasets, conduct strong scaling experiments using 1, 2, and 4 nodes. Report your results.

Experiment 3: GPU Scaling

- Using the CDS2 and CDS3 datasets, conduct strong scaling experiments using 1, 2 and 4 nodes.
    - Note: 1 node isn't big enough for the CDS3 datasets, so only run strong scaling GPU experiments with 2 and 4 nodes

Analysis:

- Analyze your strong scaling experiment results with the paper's strong scaling experiment results. Compare and contrast your experimental setups and experimental results. Finally, include your belief on whether or not your experimental results validate the claims that the paper is trying to make about scaling, memory usage, etc.

More Hints:

- If you are unable to request nodes/procs using `-l nodes=4:ppn=24...` because of coc-ice quotas, try using `-L tasks=4:lprocs=24` instead. This works for both CPU and GPU runs.
- To check if you have the correct resources requested, try running `cat $PBS_NODEFILE` to check which nodes you have, and `cat $PBS_GPUFILE` to check which GPUs you have
- Use the `-npernode <number of GPUs per node>` in your mpirun command if you are finding that multinode GPU runs only run on one node's GPU.
- For all experiments, report the totGFLOPS number for compute usage, and total bandwidth number.

Deliverables:

- Submit a report with
    - The answers to the reading questions
    - Your Experiment 1, 2, and 3 run configurations, build configurations, and results
    - Graphs for strong scaling experiments
    - Your analysis of your experimental results.
- Any other artifacts (i.e. build scripts, run scripts, data gathering/processing scripts).

Bickston Laenger | Yanni Ma MemXCT Paper Questions

1. What is XCT and how is it used in the real world?

X-ray computed tomography (XCT) is a nondestructive 3D imaging technique that is widely used to observe and understand the internal morphology of samples and materials. It works by using X-rays to create cross-sectional images of an object, which can then be reconstructed into a 3D image. In the real world, XCT is used in a variety of fields, including medicine, engineering, and materials science. For example, it can be used to study the internal structure of bones or other tissues in medical imaging, or to analyze the microstructure of materials such as metals or ceramics in materials science. Synchrotron light sources like the Advanced Photon Source (APS) can provide high-brilliance X-rays that enable tomographic imaging of centimeter-sized samples at sub-micrometer spatial resolution.

2. What is the difference between a tomogram and a sinogram?

A sinogram is a cross-sectional view of the projections of an object taken at different angles, while a tomogram is a 2D image slice that is reconstructed from the sinogram. The sinogram represents the raw data collected during the scanning process, while the tomogram is the final output that represents the internal structure of the object being scanned. The sinogram consists of Iθ measurements from f, where θ is the angle of projection and f is the object being scanned. The goal of a tomographic reconstruction algorithm is to recover a 2D image slice (tomogram) from its corresponding sinogram θ(s).

3. What are the major bottlenecks of conventional XCT approaches, and how does this paper approach the problem differently?

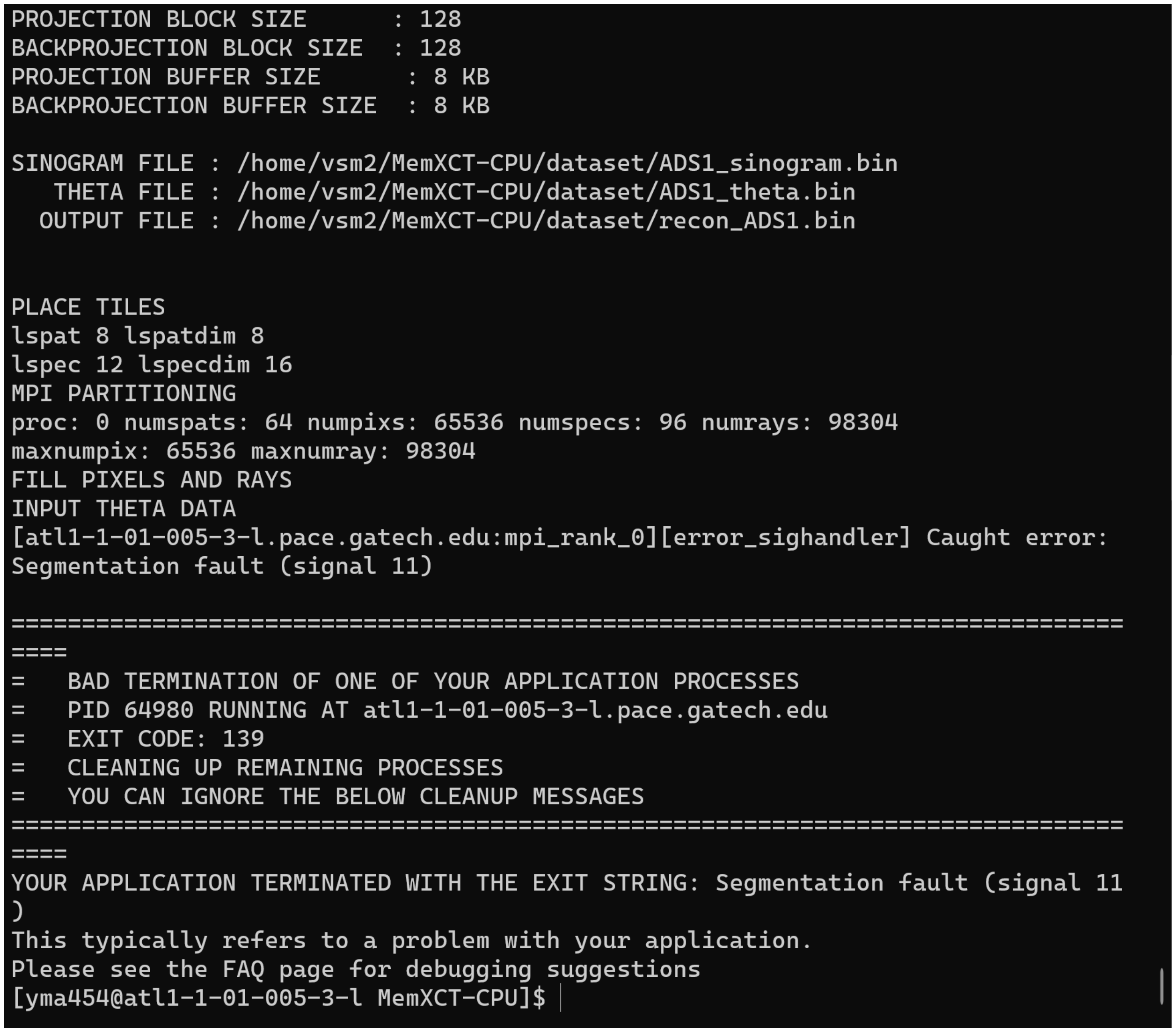
Conventional X-ray computed tomography approaches can suffer from several bottlenecks, including long acquisition times, high radiation doses, and limited resolution. This paper, MemXCT: Memory-Centric X-ray CT Reconstruction with Massive Parallelization, approaches the problem differently by using advanced iterative reconstruction techniques that can produce high-quality images from noisy measurements. The paper also uses a memory-centric approach to optimize data access patterns and minimize communication overheads, which enables efficient parallelization of the reconstruction process on modern high-performance computing systems. Additionally, the paper introduces a multi-stage input buffering technique that improves process-level data communication and locality while enabling data reuse in first-level caches and minimizing memory latency. Overall, this approach enables faster and more accurate XCT reconstructions while reducing radiation exposure and computational costs.

4. Explain two of the optimizations that MemXCT uses to improve performance.

MemXCT uses several optimizations to improve performance, two of which are the memory-centric approach and multi-stage input buffering. The memory-centric approach optimizes data access patterns and minimizes communication overheads by storing intermediate data structures in compressed format. This reduces the memory footprint and enables efficient parallelization of the reconstruction process on modern high-performance computing systems. By minimizing data movement between different levels of the memory hierarchy, MemXCT reduces the overall execution time and improves performance.

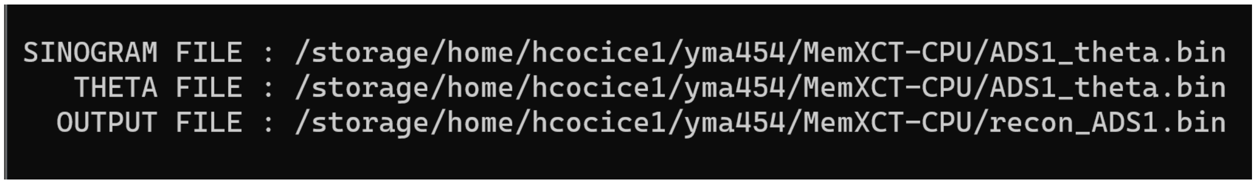
The multi-stage input buffering technique improves process-level data communication and locality while enabling data reuse in first-level caches and minimizing memory latency. This technique involves buffering input data at multiple stages of the reconstruction pipeline, which reduces the number of times that data needs to be fetched from main memory. By reducing memory access latency and improving cache utilization, this technique improves performance and enables faster XCT reconstructions. Overall, these optimizations enable MemXCT to produce high-quality images from noisy measurements while reducing radiation exposure and computational costs.

Experiment 1:


```
PROJECTION BLOCK SIZE      : 128
BACKPROJECTION BLOCK SIZE  : 128
PROJECTION BUFFER SIZE      : 8 KB
BACKPROJECTION BUFFER SIZE  : 8 KB

SINOGRAM FILE : /home/vsm2/MemXCT-CPU/dataset/ADS1_sinogram.bin
   THETA FILE : /home/vsm2/MemXCT-CPU/dataset/ADS1_theta.bin
  OUTPUT FILE : /home/vsm2/MemXCT-CPU/dataset/recon_ADS1.bin


PLACE TILES
lspat 8 lspatdim 8
lspec 12 lspecdim 16
MPI PARTITIONING
proc: 0 numspats: 64 numpixs: 65536 numspecs: 96 numrays: 98304
maxnumpix: 65536 maxnumray: 98304
FILL PIXELS AND RAYS
INPUT THETA DATA
[atl1-1-01-005-3-l.pace.gatech.edu:mpi_rank_0][error_sighandler] Caught error:
Segmentation fault (signal 11)

===============================================================================
====
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   PID 64980 RUNNING AT atl1-1-01-005-3-l.pace.gatech.edu
=   EXIT CODE: 139
=   CLEANING UP REMAINING PROCESSES
=   YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
===============================================================================
====
YOUR APPLICATION TERMINATED WITH THE EXIT STRING: Segmentation fault (signal 11
)
This typically refers to a problem with your application.
Please see the FAQ page for debugging suggestions
[yma454@atl1-1-01-005-3-l MemXCT-CPU]$
```

To fix the file paths, I made these modifications:

In the run.sh.intel file


```
SINOGRAM FILE : /storage/home/hcocice1/yma454/MemXCT-CPU/ADS1_theta.bin
   THETA FILE : /storage/home/hcocice1/yma454/MemXCT-CPU/ADS1_theta.bin
  OUTPUT FILE : /storage/home/hcocice1/yma454/MemXCT-CPU/recon_ADS1.bin
```

Output:

```
[yma454@atl1-1-02-003-19-l MemXCT-CPU]$ ./run.sh.intel
NUM. THETA           : 360
NUM. RHO             : 256
NUM. X PIXELS        : 256
NUM. Y PIXELS        : 256

NUM. OF PIXELS       : 65536
NUM. OF RAYS         : 92160

NUM. ITERATIONS      : 24

SPATIAL TILE SIZE    : 32
SPECTRAL TILE SIZE   : 32

NUMBER OF X TILES    : 8
NUMBER OF Y TILES    : 8
NUM. OF THETA TILES  : 12
NUM. OF RHO TILES    : 8

NUM. SPATIAL TILES   : 64
NUM. SPECTRAL TILES  : 96

NUM. OF X PIXELS (EXT) : 256
NUM. OF Y PIXELS (EXT) : 256
NUM. OF ANGLES (EXT)   : 384
NUM. OF RHOS (EXT)     : 256

NUM. OF PIXELS (EXT)   : 65536
NUM. OF RAYS (EXT)     : 98304

NUMBER OF PROCESSES    : 1
NUMBER OF THRD./PROC.  : 1

INTEGER: 4, FLOAT: 4, LONG: 8, SHORT: 2, POINTER: 8
APPROX. MEMORY     TOTAL: 5.273438e-01 GB
APPROX. MEMORY PER PROC.: 5.273438e-01 GB

X START       : -1.280000e+02
Y START       : -1.280000e+02
PIXEL SIZE    : 1.000000e+00
RHO START     : -1.280000e+02
RAY LENGTH    : 5.120000e+02

SPATIAL INDEXING       : 5
SPECTRAL INDEXING      : 5
 1: CARTESIAN, NATURAL
 2: CARTESIAN, TRANSPOSED
 3: MORTON, NATURAL
 4: MORTON, TRANSPOSED
 5: HILBERT
PROJECTION BLOCK SIZE      : 128
BACKPROJECTION BLOCK SIZE  : 128
PROJECTION BUFFER SIZE      : 8 KB
BACKPROJECTION BUFFER SIZE  : 8 KB

SINOGRAM FILE : /storage/home/hcocice1/yma454/MemXCT-CPU/ADS1_theta.bin
   THETA FILE : /storage/home/hcocice1/yma454/MemXCT-CPU/ADS1_theta.bin
  OUTPUT FILE : /storage/home/hcocice1/yma454/MemXCT-CPU/recon_ADS1.bin


PLACE TILES
lspat 8 lspatdim 8
lspec 12 lspecdim 16
MPI PARTITIONING
proc: 0 numspats: 64 numpixs: 65536 numspecs: 96 numrays: 98304
maxnumpix: 65536 maxnumray: 98304
FILL PIXELS AND RAYS
INPUT THETA DATA
DOMAIN PARTITIONING

CONSTRUCT PROJECTION MATRIX
CSR STORAGE: 28200745 (0.210112 GB) rownzmax 511
RAY-TRACING TIME: 5.276949e+00
```

```
CONSTRUCT BACKPROJECTION MATRIX
CSR STORAGE: 28200745 (0.210112 GB) rownzmax 524
TRANSPOSITION TIME: 5.365789e-01

BLOCKING PROJECTION MATRIX
NUMBER OF BLOCKS: 720 BUFFSIZE: 2048
NUMBER OF BUFFERS: 1954 AVERAGE BUFF/BLOCK: 2.713889 MAX BUFF/BLOCK: 4
BUFFER MAP: 3198686 (0.011916 GB)
CSR STORAGE: 28200745 (0.157584 GB) buffnzmax: 265 STORAGE EFFICIENCY: 1.500000 DATA REUSE:
8.816353
BLOCKING TIME: 1.459359e+00
BLOCKING BACKPROJECTION MATRIX
NUMBER OF BLOCKS: 512 BUFFSIZE: 2048
NUMBER OF BUFFERS: 1468 AVERAGE BUFF/BLOCK: 2.867188 MAX BUFF/BLOCK: 3
BUFFER MAP: 2642958 (0.009846 GB)
CSR STORAGE: 28200745 (0.157584 GB) buffnzmax: 283 STORAGE EFFICIENCY: 1.500000 DATA REUSE:
10.670145
BLOCKING TIME: 1.492284e+00

FILL PROJECTION MATRIX
TIME: 3.203518e+00
FILL BACKPROJECTION MATRIX
TIME: 3.611972e+00
REDUCTION MAPPINGS
PREPROCESSING TIME: 1.558394e+01
INPUT MEASUREMENT DATA
INPUT ENDS
GRADIENT-DESCENT OPTIMIZATION
iter: 0 error: 1.179422e+03 gradnorm: 3.694423e+05
iter: 1 error: 1.173446e+03 gradnorm: 4.636519e+04
iter: 2 error: 1.171401e+03 gradnorm: 1.897963e+04
iter: 3 error: 1.169977e+03 gradnorm: 1.264689e+04
iter: 4 error: 1.168980e+03 gradnorm: 6.982585e+03
iter: 5 error: 1.168076e+03 gradnorm: 4.947881e+03
iter: 6 error: 1.167213e+03 gradnorm: 1.231177e+04
iter: 7 error: 1.167299e+03 gradnorm: 3.714061e+03
iter: 8 error: 1.166516e+03 gradnorm: 2.614192e+03
iter: 9 error: 1.165974e+03 gradnorm: 1.755739e+03
iter: 10 error: 1.165865e+03 gradnorm: 1.541367e+03
iter: 11 error: 1.165573e+03 gradnorm: 2.857313e+03
iter: 12 error: 1.165446e+03 gradnorm: 1.452238e+03
iter: 13 error: 1.165451e+03 gradnorm: 1.607519e+03
```

```
iter: 14 error: 1.165322e+03 gradnorm: 1.067477e+03
iter: 15 error: 1.165187e+03 gradnorm: 8.296239e+02
iter: 16 error: 1.165090e+03 gradnorm: 6.705502e+02
iter: 17 error: 1.164947e+03 gradnorm: 7.662736e+02
iter: 18 error: 1.164855e+03 gradnorm: 1.248302e+03
iter: 19 error: 1.164841e+03 gradnorm: 7.109590e+02
iter: 20 error: 1.164889e+03 gradnorm: 4.571436e+02
iter: 21 error: 1.164722e+03 gradnorm: 3.383582e+02
iter: 22 error: 1.164638e+03 gradnorm: 3.454229e+02
iter: 23 error: 1.164671e+03 gradnorm: 9.108458e+02
iter: 24 error: 1.164517e+03 gradnorm: 5.214784e+02
recon: 9.818047e+00 proj: 6.444469e+00 (6.410794e+00 2.473593e-03 3.118992e-02) backproj:
3.318175e+00 (7.747889e-03 9.460449e-04 3.309477e+00) ctime: 5.479670e-02 wtime: 3.378391e-04
numproj: 49 numback: 25
proj: 6.410794e+00 s (0.431097 GFLOPS) backproj: 3.309477e+00 s (0.426060 GFLOPS) avGFLOPS:
0.429382 totGFLOPS: 0.429382
proj: 1.391085 GB/s back: 1.358042 GB/s av: 1.379835 GB/s tot: 1.379835 GB/s
```

Total Time: 2.613116e+01

ULTIMATELY THIS NEEDS TO BE AVERAGED, GRAPHED, AND COMPARED

Run.sh.intel:

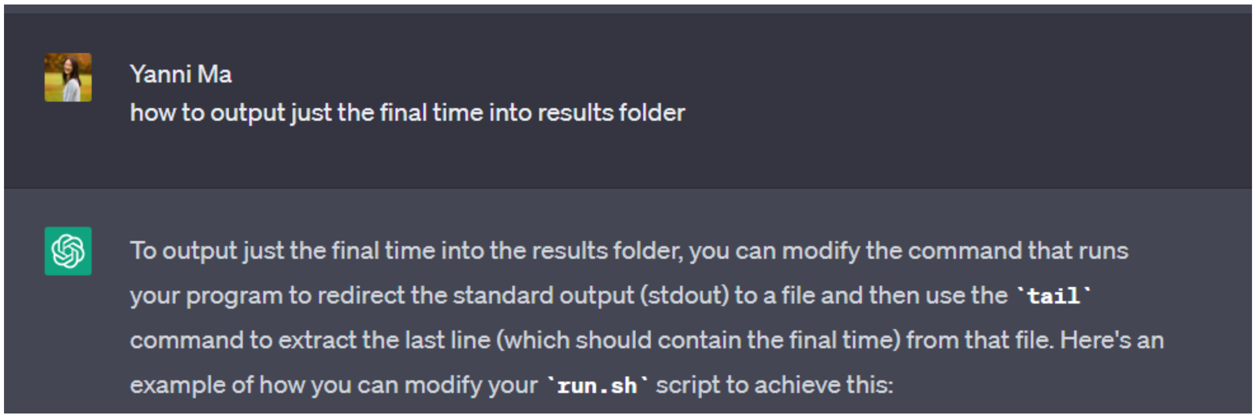
```
[yma454@atl1-1-02-003-19-l MemXCT-CPU]$ cat run.sh.intel
#!/bin/bash

#DOMAIN INFORMATION
export NUMTHE=360
export NUMRHO=256
export PIXSIZE=1
#SOLVER DATA
export NUMITER=24
#TILE SIZE (MUST BE POWER OF TWO)
export SPATSIZE=32
export SPECSIZE=32
#BLOCK SIZE
export PROJBLOCK=128
export BACKBLOCK=128
#BUFFER SIZE
export PROJBUFF=8
export BACKBUFF=8
#I/O FILES
export THEFILE=/storage/home/hcocice1/yma454/MemXCT-CPU/ADS1_theta.bin
## MODIFICATIONS MADE HERE
export SINFILE=/storage/home/hcocice1/yma454/MemXCT-CPU/ADS1_theta.bin
export OUTFILE=/storage/home/hcocice1/yma454/MemXCT-CPU/recon_ADS1.bin

export OMP_NUM_THREADS=1

mpirun -np 1 ./memxct
```

Automate Experiment: Running into errors though

```
#!/bin/bash

#DOMAIN INFORMATION
export NUMTHE=360
export NUMRHO=256
export PIXSIZE=1
#SOLVER DATA
export NUMITER=24

#I/O FILES
export THEFILE=/storage/home/hcocice1/yma454/MemXCT-CPU/ADS1_theta.bin
export SINFILE=/storage/home/hcocice1/yma454/MemXCT-CPU/ADS1_theta.bin
export OUTFILE=/storage/home/hcocice1/yma454/MemXCT-CPU/recon_ADS1.bin

export OMP_NUM_THREADS=1

# define the different configurations to test
tile_sizes=(32 64)
block_sizes=(64 128)
```

```
buffer_sizes=(4 8)
compilers=("gcc" "intel")
mpi_libraries=("openmpi" "mvapich2")

# create a results directory
mkdir results

# run experiments with different configurations
for tile_size in "${tile_sizes[@]}"; do
  for block_size in "${block_sizes[@]}"; do
    for buffer_size in "${buffer_sizes[@]}"; do
      for compiler in "${compilers[@]}"; do
        for mpi_library in "${mpi_libraries[@]}"; do

          # compile the program with the current configuration
          make clean
          make COMPILER=$compiler MPI_LIBRARY=$mpi_library \
              SPATSIZE=$tile_size SPECSIZE=$tile_size \
              PROJBLOCK=$block_size BACKBLOCK=$block_size \
              PROJBUFF=$buffer_size BACKBUFF=$buffer_size

          # run the program and record the performance metrics
          for i in {1..3}; do

output_file="results/memxct_${tile_size}_${block_size}_${buffer_size}_${compiler}_${mpi_library}_${i}.t
xt"
            mpirun -np 1 ./memxct > $output_file
          done
        done
      done
    done
  done
done
```

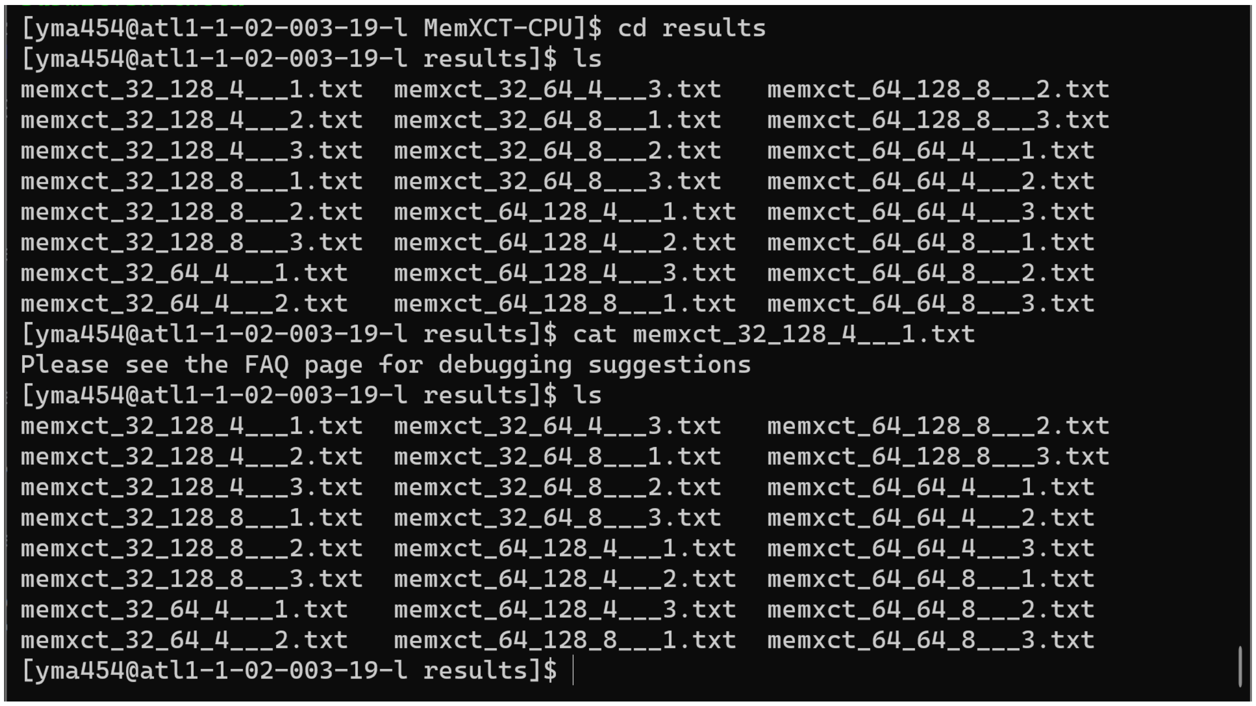
Determining how to extract just the final time into the results:


Yanni Ma
how to output just the final time into results folder

To output just the final time into the results folder, you can modify the command that runs your program to redirect the standard output (stdout) to a file and then use the `tail` command to extract the last line (which should contain the final time) from that file. Here's an example of how you can modify your `run.sh` script to achieve this:

So I tried to automate it and then either append last line into one file or last line into all these files (see below)

It is not quite working but I think I will get it to work soon


```
[yma454@atl1-1-02-003-19-l MemXCT-CPU]$ cd results
[yma454@atl1-1-02-003-19-l results]$ ls
memxct_32_128_4___1.txt  memxct_32_64_4___3.txt   memxct_64_128_8___2.txt
memxct_32_128_4___2.txt  memxct_32_64_8___1.txt   memxct_64_128_8___3.txt
memxct_32_128_4___3.txt  memxct_32_64_8___2.txt   memxct_64_64_4___1.txt
memxct_32_128_8___1.txt  memxct_32_64_8___3.txt   memxct_64_64_4___2.txt
memxct_32_128_8___2.txt  memxct_64_128_4___1.txt  memxct_64_64_4___3.txt
memxct_32_128_8___3.txt  memxct_64_128_4___2.txt  memxct_64_64_8___1.txt
memxct_32_64_4___1.txt   memxct_64_128_4___3.txt  memxct_64_64_8___2.txt
memxct_32_64_4___2.txt   memxct_64_128_8___1.txt  memxct_64_64_8___3.txt
[yma454@atl1-1-02-003-19-l results]$ cat memxct_32_128_4___1.txt
Please see the FAQ page for debugging suggestions
[yma454@atl1-1-02-003-19-l results]$ ls
memxct_32_128_4___1.txt  memxct_32_64_4___3.txt   memxct_64_128_8___2.txt
memxct_32_128_4___2.txt  memxct_32_64_8___1.txt   memxct_64_128_8___3.txt
memxct_32_128_4___3.txt  memxct_32_64_8___2.txt   memxct_64_64_4___1.txt
memxct_32_128_8___1.txt  memxct_32_64_8___3.txt   memxct_64_64_4___2.txt
memxct_32_128_8___2.txt  memxct_64_128_4___1.txt  memxct_64_64_4___3.txt
memxct_32_128_8___3.txt  memxct_64_128_4___2.txt  memxct_64_64_8___1.txt
memxct_32_64_4___1.txt   memxct_64_128_4___3.txt  memxct_64_64_8___2.txt
memxct_32_64_4___2.txt   memxct_64_128_8___1.txt  memxct_64_64_8___3.txt
[yma454@atl1-1-02-003-19-l results]$
```

Automate it - script - needs some work but is almost there:

```
[yma454@atl1-1-02-003-19-l MemXCT-CPU]$ cat run.sh.intel
#!/bin/bash

#DOMAIN INFORMATION
```

```
export NUMTHE=360
export NUMRHO=256
export PIXSIZE=1
#SOLVER DATA
export NUMITER=24
#TILE SIZE (MUST BE POWER OF TWO)
#export SPATSIZE=32
#export SPECSIZE=32
#BLOCK SIZE
#export PROJBLOCK=128
#export BACKBLOCK=128
#BUFFER SIZE
#export PROJBUFF=8
#export BACKBUFF=8
#I/O FILES
export THEFILE=/storage/home/hcocice1/yma454/MemXCT-CPU/ADS1_theta.bin
export SINFILE=/storage/home/hcocice1/yma454/MemXCT-CPU/ADS1_theta.bin
export OUTFILE=/storage/home/hcocice1/yma454/MemXCT-CPU/recon_ADS1.bin

export OMP_NUM_THREADS=1

# define the different configurations to test
tile_sizes=(32 64)
block_sizes=(64 128)
buffer_sizes=(4 8)
compilers=("gcc" "intel")
mpi_libraries=("openmpi" "mvapich2")

# create a results directory
mkdir results

# run experiments with different configurations

## this is supposed to use for loops to loop over each config and run it
for tile_size in "${tile_sizes[@]}"; do
  for block_size in "${block_sizes[@]}"; do
    for buffer_size in "${buffer_sizes[@]}"; do
      #for compiler in "${compilers[@]}"; do
       # for mpi_library in "${mpi_libraries[@]}"; do


         # compile the program with the current configuration
        #  make clean
        # make COMPILER=$compiler MPI_LIBRARY=$mpi_library \
            SPATSIZE=$tile_size SPECSIZE=$tile_size \
            PROJBLOCK=$block_size BACKBLOCK=$block_size \
            PROJBUFF=$buffer_size BACKBUFF=$buffer_size

        # run the program and record the performance metrics
        for i in {1..3}; do

      tmp_file="results/memxct_${tile_size}_${block_size}_${buffer_size}_${compiler}_${mpi_library}
      _${i}_tmp.txt"

      output_file="results/memxct_${tile_size}_${block_size}_${buffer_size}_${compiler}_${mpi_librar
      y}_${i}.txt"
          mpirun -np 1 ./memxct > $tmp_file
          tail -n 1 $tmp_file >> $output_file
```
## **this is supposed to append just the last line to output file I think I need to change OUTFILE ABOVE TOO**
```
          rm $tmp_file
        done
      done
    done
  #done
 # done
done
[yma454@atl1-1-02-003-19-l MemXCT-CPU]$
```