Parallel_AWT

Milestone Report

URL: https://hakuz-y.github.io/Parallel_AWT/milestone/

Updated Schedule

Deadline	Task	Partner
4/16	Improve the sequential implementation (inplace details)	Maggie
4/17	Optimize the shared memory model	Youheng
4/18	Further optimize the algorithm	Maggie
4/19	Finish current TODOs in the baseline and shared memory version	Youheng
4/20	Transition to MPI implementation	Together
4/21	Optimize MPI to achieve the targeting speedup (4-5x with 8 threads)	Maggie
4/22	Experiment with PSC if we have the access, achieve the targeting speedup	Youheng
4/23	If the above is done by 20th, try to optimize both version with a perfect speedup (10-11x with 12 threads)	Together
4/24	If we were really lucky and achieve all of the above early, try to implement the lock-free version	Together
4/25	Profiling, Collecting speedup graphs	Together
4/27	draft the final report, poster session preparation	Together
4/28	Wrap up, final review on the report and poster session presentation	Together
4/29	Presentation	Together

Work Completed So Far

For the first 2 weeks, we have completed the Open_MP version of parallelizing the 2D multi-level Adaptive Wavelet Transformation that processes image compression. We learned the algorithm from the paper Adaptive Wavelet Transforms via Lifting and implemented a baseline from it, mainly featuring the algorithm's adaptive nature and the lifting scheme.

We implemented the lifting scheme approach of wavelet transform, which involves 3 stages:

- splitting the input signal into even and odd samples
- predicting the odd samples using neighboring even samples, where the prediction error becomes the detail coefficient
- updating the even samples using hte detail coefficients to maintain certain properties of the signal

To make the transformation adaptive, we dynamically select the filter for each sample based on minimizing the local prediction error, which allows for a better representation of signal structure (e.g., edges).

Once the baseline was completed, we used Open_MP to parallelize it under a shared memory model. We used directives such as loop construct during the row-wise and column-wise stages of the transformation as they are nested for loops. We measured transformation time and achieved a 4-5x speedup with 8 threads and 6-7x speedup with 12 threads. We also created the metrics with MSE (Mean Squared Error) and SSIM (Structural similarity index measure) to evaluate the quality of the compressed image. The detailed results are in the section `Preliminary results` .

There 2 different ways to handle the horizontal, vertical and diagnal coefficients details after the AWT: store in the image inplace or in separate matrixes. We have tried both deisgn and compared the speedup result to see which approach has a better memory access pattern and better for parallelizing, we chose to use separate stores for now, but we need to do more experiments before the final decision.

Goals and Deliverables

Currently, we followed our schedule and goals well in the proposal. In the first week, we successfully implemented a baseline of image compression by using a 2D multi-level Adaptive Wavelet Transformation and Reconstruction with the following steps: 1D discrete wavelet transformation –> 2D DWT –> 2D multi-level DWT –> 1D AWT –> 2D AWT –> 2D multi-level AWT. In the second week, we created 3 test dataset: easy_2048, medium_8192, hard_16384. Since the largest test file has a size of 1.6GB, we didn't push it into our repo, but included the original image and the script to generate the grayscale txt. We also created 2 metrics, MSE and ISSM to test the correctness of our implementation, which should be 0 and 1 respectively for a threshold of 0.

Finally, we parallelized the transformation and reconstruction by using the Open_MP, acheving a 4-5x speedup with 12 threads.

We believe we could finish the next deliverable that uses MPI model. Though we are not fully confident to achieve an optimal speedup as the shared memory model due to the communication complexity of AWT, we will try to optimize it as much as possible. similarly, the perfect speedup in the “nice to haves” section of the proposal might be hard to achieve, but we will use different strategies to see the different optimization results and analyze them as well.

Plan for poster session

We plan to mainly provide a brief explanation of the algorithm, a walkthrough of our implementation and approaches to parallelize it, as well as speedup analysis graphs at the poster session. We can also demo compressing a photo of reasonable size live during the poster session.

- Visualizations:
  - A simple demo compares the image compression speed between a sequential, shared memory model, and MPI model.
  - Some plots shows the speedup and metrics of 2 different parallelism with different levels and threshold.
- Contexs:
  - Some main conclusion in our projects.
  - What we learned from our sources, and what we learned during the implementation and optimization.
  - Some challenges and limitations we met during the implementation.

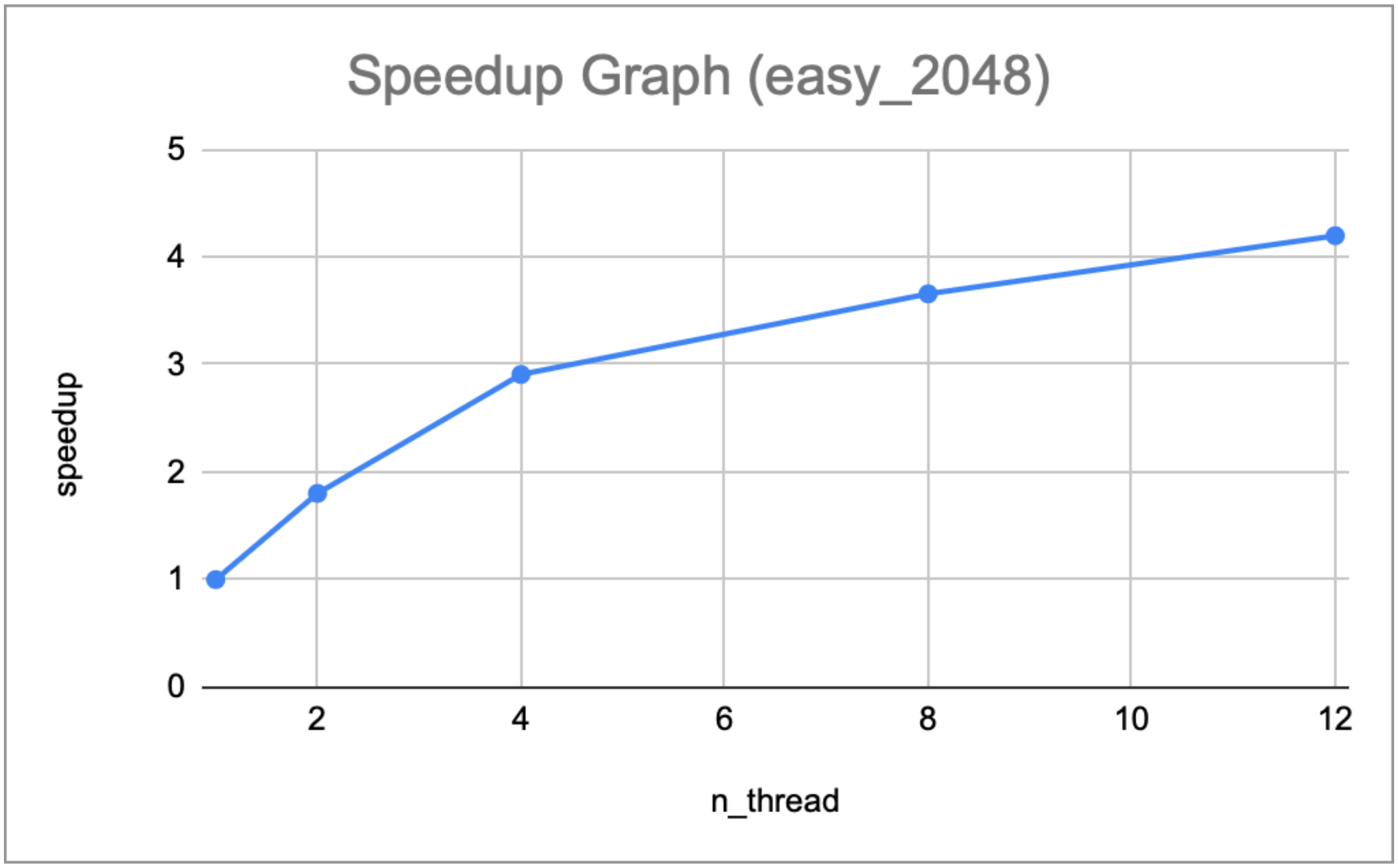
Preliminary results

We have started parallelizing the algorithm using a shared memory model with OpenMP. All of the graphs are based on the total computation time, which includes both transformation and reconstruction. Detailed timing results can be found in the spreadsheet via this link.

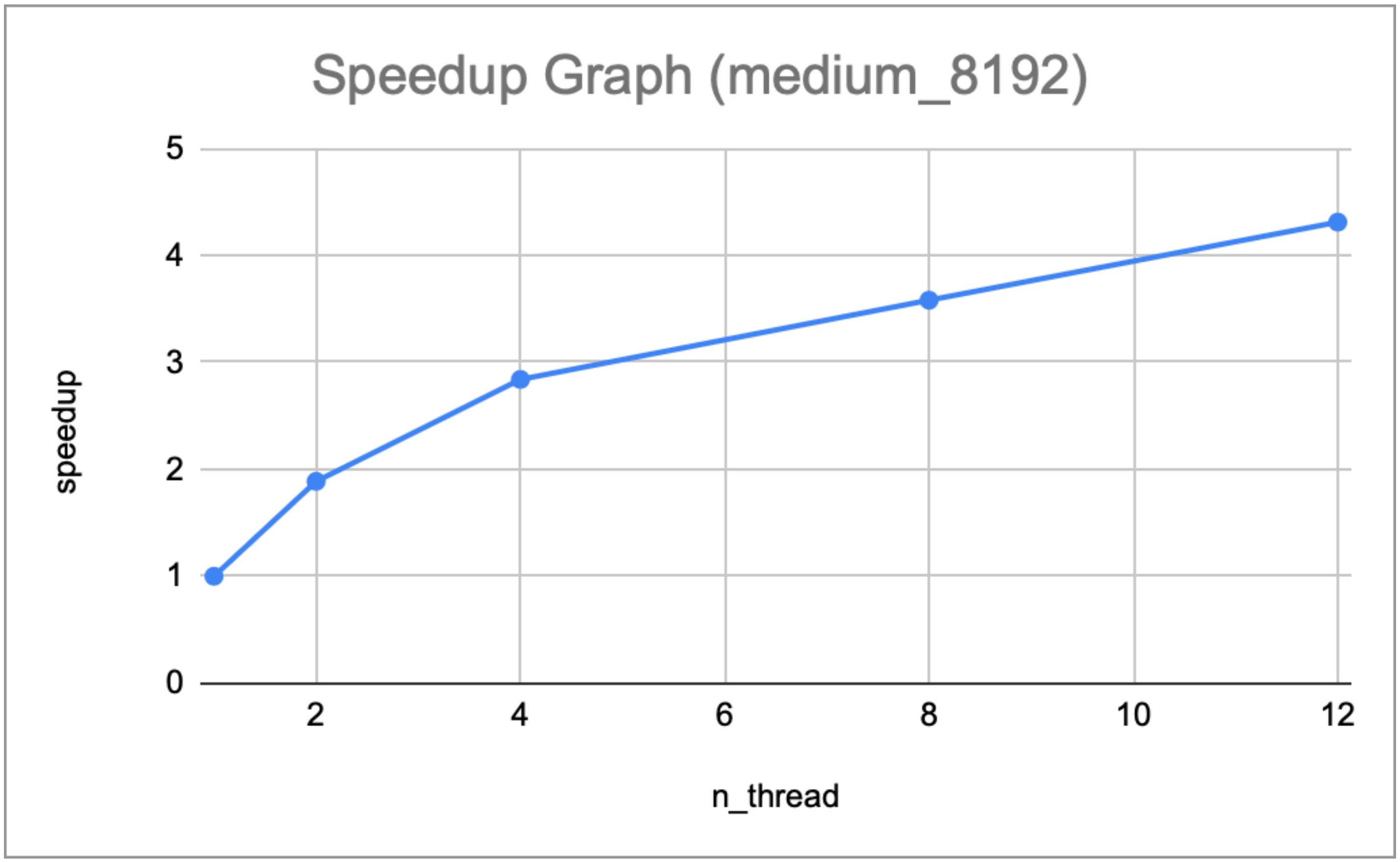
All curves demonstrate a consistent upward trend as the number of threads increases, which means our implementation scales well so far, and that the per-thread workload is balanced. This also suggests that the 12 available cores in the current system haven’t saturated the full parallel potential of the program.

We have yet to find the threshold when the speedup would start to taper off or degrade, which would need to be done on the GHC or PSC machines. We would also likely to be expecting a different speedup trend and scaling behavior once switching to an MPI model due to the different communication pattern and resulting overhead.

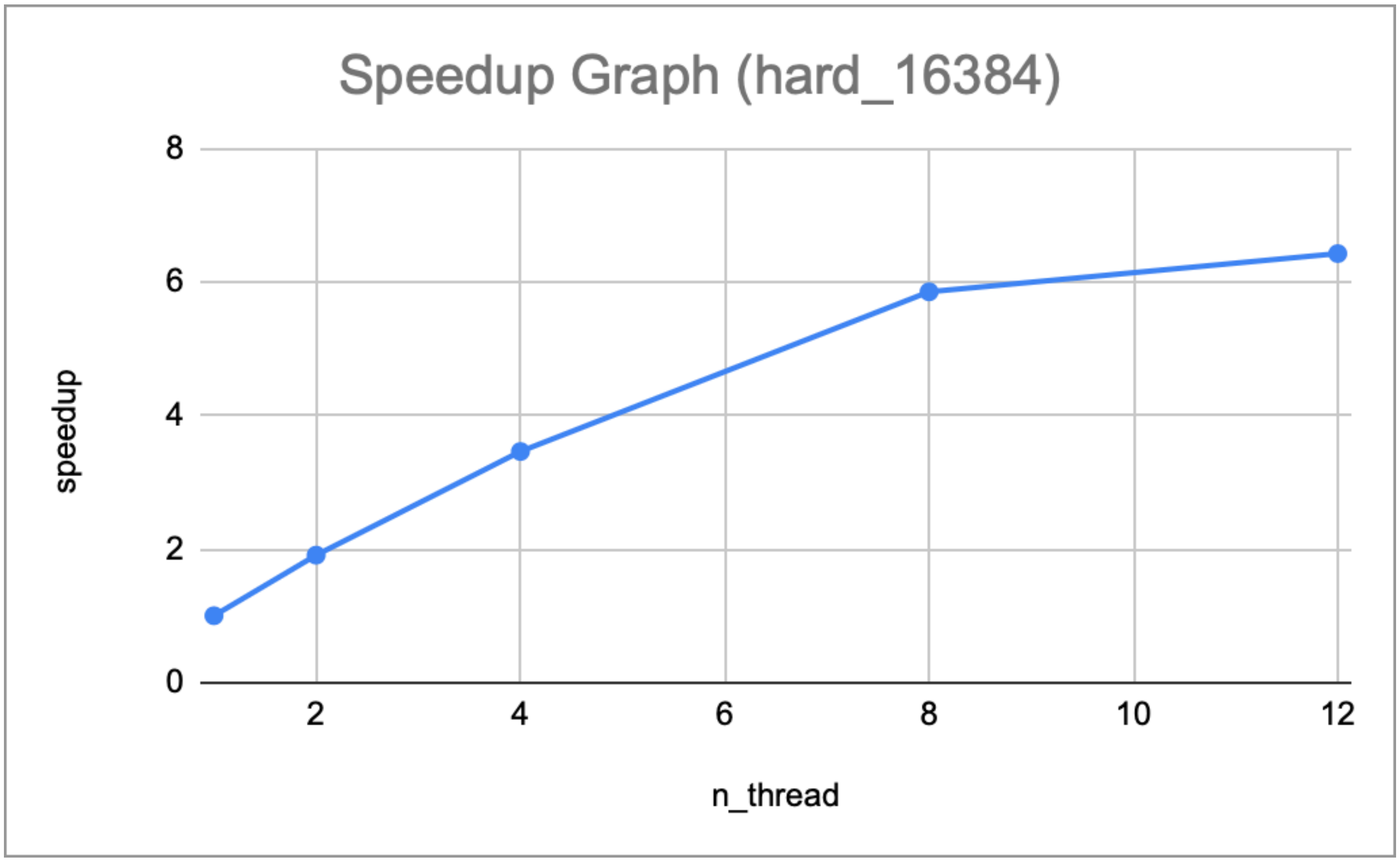
> Total speedup for easy_2048



> Total speedup for medium_8192



> Total speedup for hard_16384



Concerns

With the complexity of the algorithm and parallelizing the AWT, we are not very confident to achieve a similar speed up with MPI version, as there would be a lots of design and communication overhead. We will try different orchestrations, and some possible papers or discussions in public to see if we could find some good intuition behind it.

Another difficulty is the size of the test file, which is about 1.6 GB for the hard case. We were able to test it with at most 12 threads in our local computers, but we are not sure if we have access to the PSC, and if we have enough volume for the test files.

The rest of the work would be tuning and fixing the algorithm itself to ensure correctness while trying to achieve the targeting speedup.