Parallel 2D Multi-level Adaptive Wavelet Transformation

Maggie Gong & Youheng Yang

URL:

https://hakuz-y.github.io/Parallel_AWT/proposal/

Summary:

This project aims to implement a parallel version of the Adaptive Wavelet Transformation inspired by the paper with both shared memory model and MPI. AWT dynamically selects wavelet bases tailored to local image features, aiming to improve compression and rendering performance over traditional fixed-basis wavelet transforms. The primary objective is to enhance computational performance while preserving scalability and output quality. This involves addressing the unique challenges associated with parallelizing adaptive algorithms, such as load imbalance, data dependencies, and irregular memory access patterns.

Background:

Wavelet transforms are used for analyzing data where features vary over different scales in signal and image processing for tasks such as compression, denoising, and multi-resolution analysis. Real-world signals often have smooth regions interrupted by abrupt changes, so rapidly decaying wave-like oscillation are being used to represent such data.

To make it more interesting in parallelizing, Adaptive Wavelet Transform (AWT) dynamically selects wavelet basis functions based on local image characteristics, such as texture, edges, or smoothness. This adaptivity increases representation efficiency but introduces significant workload imbalance and data dependencies:

- Computation in one region can influence the basis function selection in adjacent regions.
- This inter-regional dependency complicates parallel execution, as decisions are no longer strictly local.

Adaptive Wavelet Transform Overview:

- Image Analysis: The image is divided into blocks, and each block is analyzed to identify characteristics such as edges, textures, and smooth regions.
- Basis Function Selection: Based on the analysis, the most suitable wavelet basis function is selected for each block to best represent its features.
- Transformation: The selected wavelet transform is applied to each block, resulting in coefficients that are more efficiently compressed due to their alignment with local image properties.

In addition to parallelism, we also plan to explore various strategies for memory access, including in-place and out-of-place implementations; with the former one having less memory overhead but higher complexity and the latter one having more straightforward parallelization but causing extra memory costs.

The Challenge:

Workload Imbalance:

- The main challenge in parallelizing AWT is the load balancing, particularly for spatially inhomogeneous problems. The distribution of data points varies significantly at each resolution level, making static partitioning inefficient.
- Existing parallel algorithms designed for non-adaptive wavelet transforms are unsuitable here, as they rely on stage-wise synchronization at each resolution level—an approach that suffers from poor data locality and load imbalance in adaptive settings.

Data Dependencies:

- Due to spatial overlap of wavelet coefficients, a single input sample at location x contributes to multiple coefficients across different resolution levels. This leads to write conflicts when multiple threads attempt to update overlapping coefficients concurrently.
- In 2D Multi-level AWT, each transformation level depends on coefficients from the previous level. This introduces sequential dependencies where level N cannot begin until level N-1 completes, potentially becoming a bottleneck.

Memory access characteristics

- The computation involves irregular memory access patterns, with non-contiguous writes during coefficient generation. This reduces cache efficiency and increases memory latency.
- Poor spatial locality due to adaptive sampling further degrades cache performance.

Data Movements:

- The algorithm requires substantial data shuffling between resolution levels, especially in the multi-level 2D case. These movements are both expensive and complex, posing additional challenges for parallelization.

Resources:

Adaptive Wavelet Transformation \
Adaptive Wavelet Rendering

It would be really helpful if we could have access of PSC machines and test beyond 8 threads.

Goals and Deliverables:

Plan to achieve:

1. Develop a fully functional sequential implementation of AWT application (Image Compression/Rendering).
2. Parallelize the application using a shared memory model (OpenMP) and distributed memory model (MPI) in C++.
3. Profile and analyze the performances of the application, targeting a 4 to 5x speedup with 8 threads.

4. Establish benchmarks to evaluate speedup and scalability across different architectures and input sizes.
5. Document our approach, design decisions, performance findings, and overall progress in detail.

Hope to achieve:

1. Achieve an ideal speedup (7 to 8x with 8 threads).
2. If time permits and progress goes smoothly, we want to explore transactional memory and lock-free techniques for synchronization, comparing their impact on performance and scalability with that of traditional locking-based implementations.

We hope by the end of this project, we develop a deeper understanding of the challenges in parallelizing a highly complex sequential algorithm , as well as the key roles that memory access pattern have in affecting performances of a parallel program.

Platform Choice:

- For CPU parallelization, we plan to use GHC and PSC machines for testing, with number of processes ranging from 1 to 128. We will start with implementing the parallel version using OpenMP with shared memory model for ease of development and then transition to MPI to improve scalability across larger number of processes.
- We will use C++ as our programming language given its performance for efficient low-level control and suitability

Schedule:

- Week 1 (Mar 26th): complete the proposal, brainstorm idea, finalize project outline
- Week 2 (April 2nd): implement the sequential C++ version of the algorithm, create test cases
- Week 3 (April 15th): initial parallel implementation, create more comprehensive test cases
- Week 4 (April 22nd): milestone report, transition from shared memory model to MPI, profiling
- Week 5 (April 28th): more profiling, data analysis and visualization, improvements, prepare final report
- Week 5 (April 29th): presentation