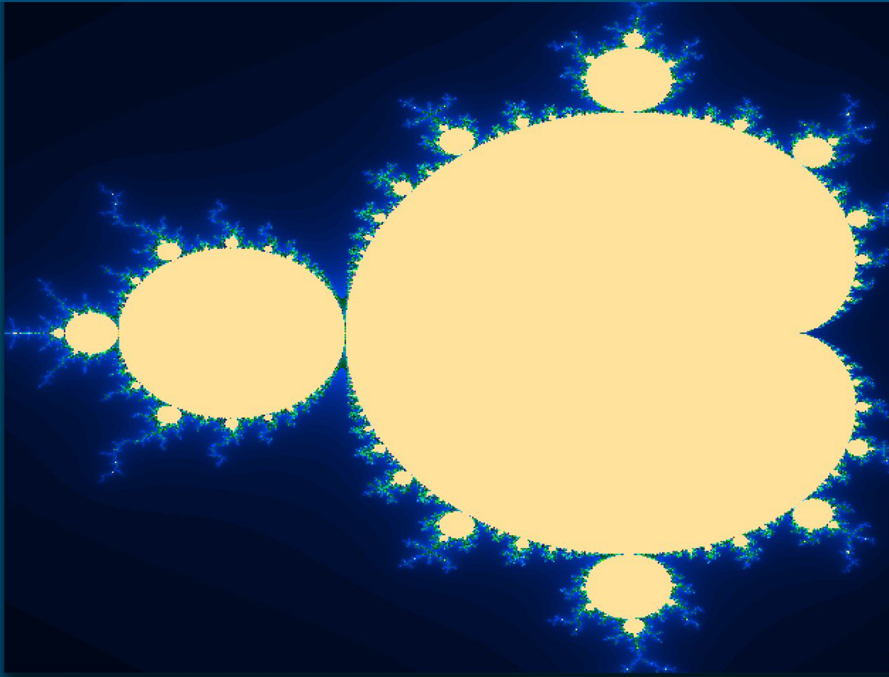
Parallelizing Mandelbrot

Group 8:
Patrick Leiser,
Jared Pugh,
Catherine Yaroslavtseva

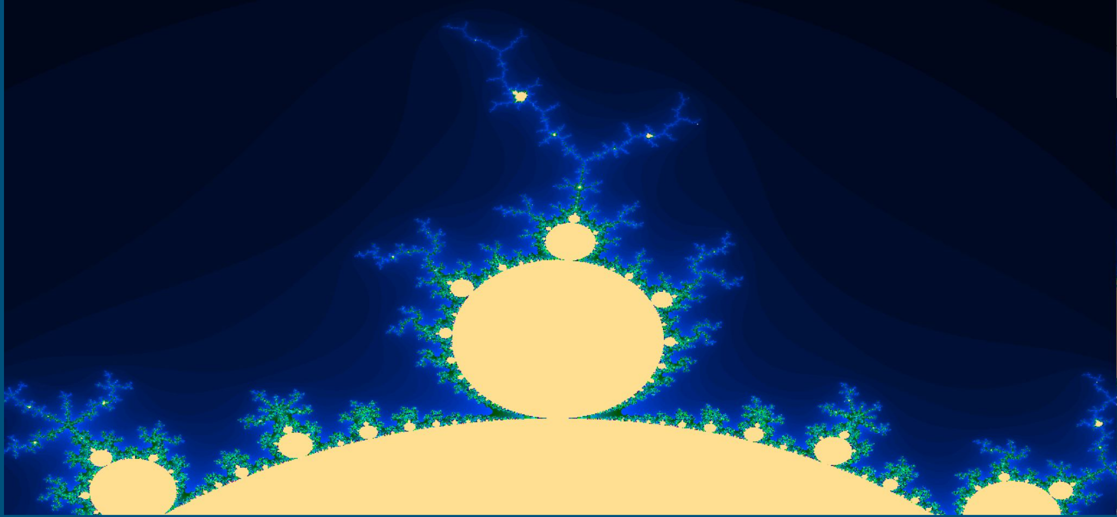
Overview

For improving the performance of our mandelbrot set calculations, we used a variety of strategies, including

- Adjusting computation distribution
    - Per-core distribution of workload
        - (OpenMP or CUDA)
    - between CPU and GPU
        - (let CPU contribute with idle time as GPU calculation progresses)
- Caching results
    - Useful when moving the viewing window, especially with horizontal/vertical shifts



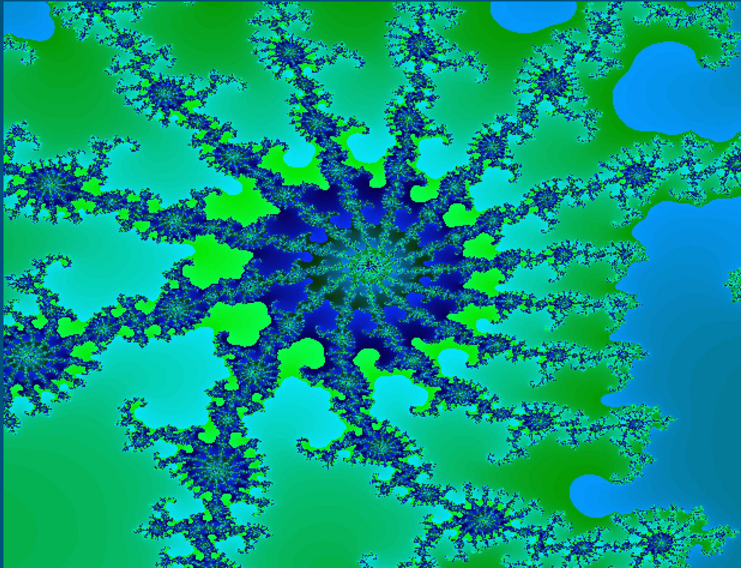
Distribution



- Sample code - block distribution
  - Unbalanced workload
    - Depending on the iterations required on a given row, some take longer to calculate
  - Balance with dynamically-scheduled distribution
    - Accomplished through dynamic scheduling in OpenMP
    - Because number of iterations for a given coordinate is hard to predict, a cyclic distribution doesn’t help much
    - Thus dynamic scheduling performs much better than either cyclic or block distributions

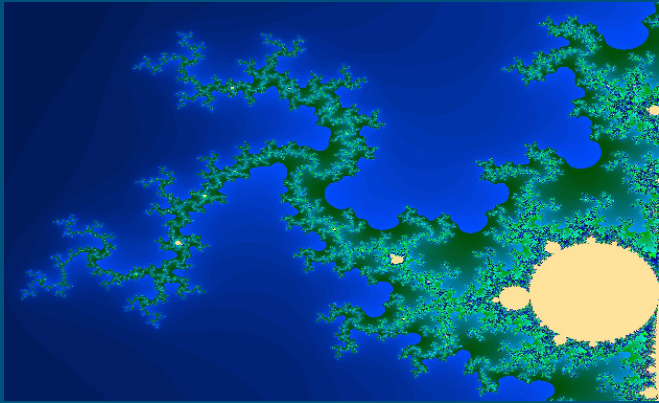
CPU vs. GPU Balancing

- Timesave from GPU is nonexistent with small problem size
  - As problem size increases, GPU starts to become more useful
  - A hybrid approach using both GPU and CPU computing can be faster than either alone
  - Calibrate a balance between GPU and CPU workload
    - Shift to GPU as workload increases
  - CPU also able to get bigger time-saves from caching
    - How colors are stored
    - And more details on next slide…



Cache

- Default implementation recalculates colors for every transformation
  - Results in redundant calculations
    - x/y values that have already been solved
    - Color values for previously calculated iterations
- Two potential cache implementations
  - Map colors to pixels indices
    - Results in significant time savings after the initial render
    - O(1) cache access time with hashmap
  - Map colors to iteration values
    - May help especially with frequently encountered colors
    - Limited speedup because we use a simple (fast) color calculation
      - Just multiplication and division, no sine function like some examples use



Next Steps

- Process constant amounts of iterations and cache the results to produce a progressive drawing
- Dynamically choose distribution between CPU and GPU
  - Have them scan from opposite directions, meet in the middle
- Enable caching for the reflection and zoom features
- Preemptively pre-caching nearby out-of-frame values
- Utilize multiple GPUs?
- Combine all our optimizations