Chris Jordan
Victor Janmey
Raymond Ko

Parallelizing K-Means for Image Segmentation

Problems and Applications

- Group points to clusters
- Fixed number of clusters (k)
- Machine learning
  - Unsupervised learning methods
  - Dividing data for more specific classifiers
  - Selection of examples for k-nearest neighbors
- Computer vision
  - Image segmentation
  - Estimating object boundaries
  - Cue for object recognition
- Network Analysis
  - Finding communities

K-means Algorithm

- Step 1: Initialization
  - Choose initial centers
    - Naïve approach
    - K-means++ (Significant speedup)
- Step 2: Iteration
  - Assign particle to centers (distance measure)
  - Re-compute cluster centers (means of points)
  - Terminate when assignments converge

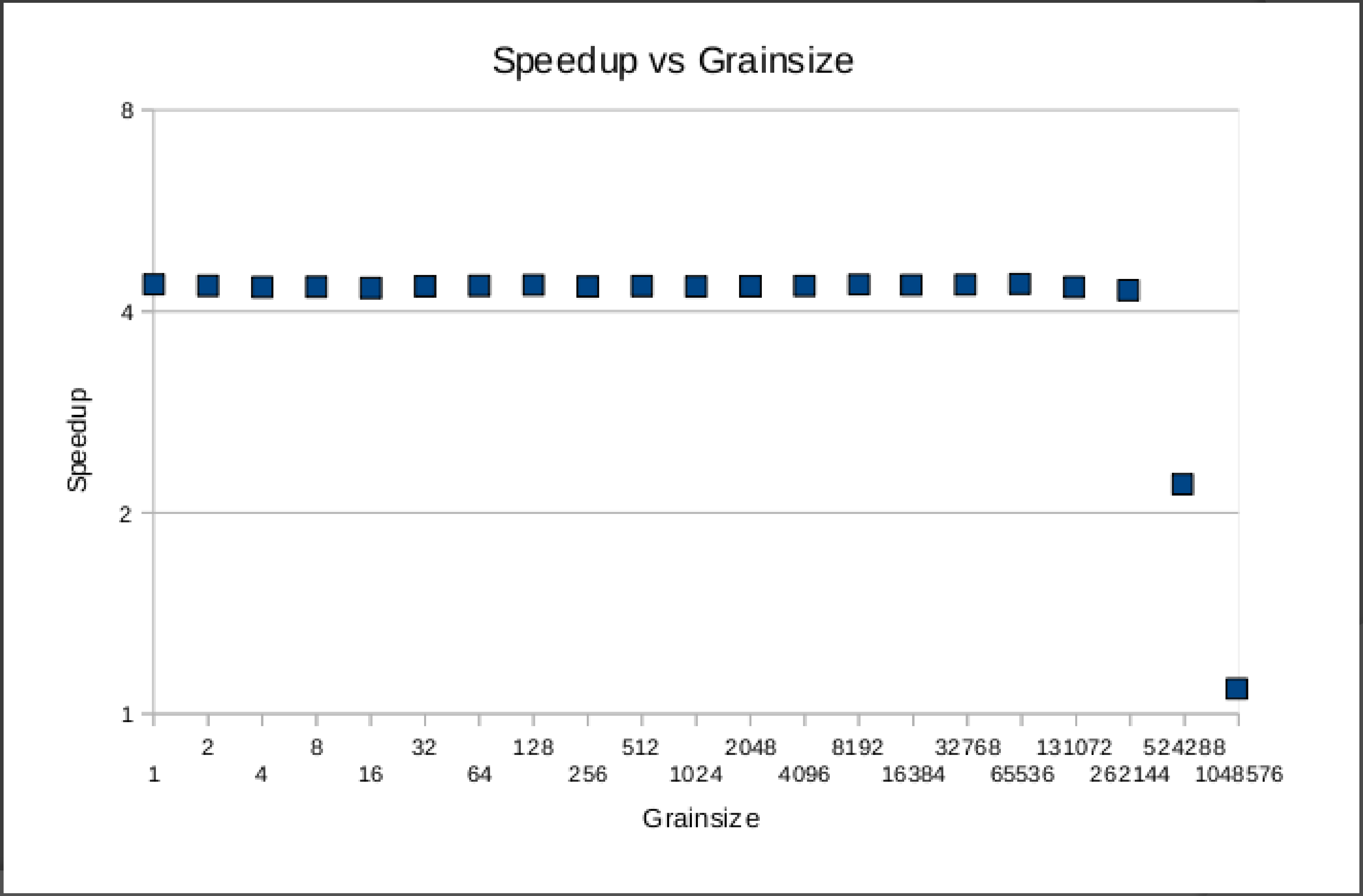
Implementation

- C++ and OpenCV image library
  - Required for TBB, OpenCV has easy image manipulation, input, and output
- Parallel: Add TBB constructs to serial
- Parallel cluster assignment: parallel_for
- Parallel cluster center calculation: parallel_reduce

Preliminary results

- Sample run: 1,024,000 pixels, 3 dimensions (HSV)
- 50 Iteration, 50 clusters
- Speedup: 4.4, Grain Size: 64,000
- Step 1 speedup: 4.42, grain size: 64,000
- Step 2 speedup: 2.89 grain size: 8,192
- Step 1 dominates since is it is (k*n) vs. O(n) for Step 2
- Similarly for k-means++ initial cluster selection, Step 1 (min distance to a cluster) is the dominating factor since it is also O(n)

Speedup vs. Grainsize



Next steps

- Investigate if similar performance is achievable with OpenMP
- Experiments
  - Speedup vs. Number of Particles
    - Is there an optimal image size given today's technology that balances speed and benefits?
  - Speedup vs. Number of Cores
    - Does it scale well? When will performance gains be apparent?
  - Speedup vs. Number of Clusters
    - At what point does it become inefficient to use the algorithm, if any?
- Algorithm improvements
  - K-means++
  - Triangle inequality