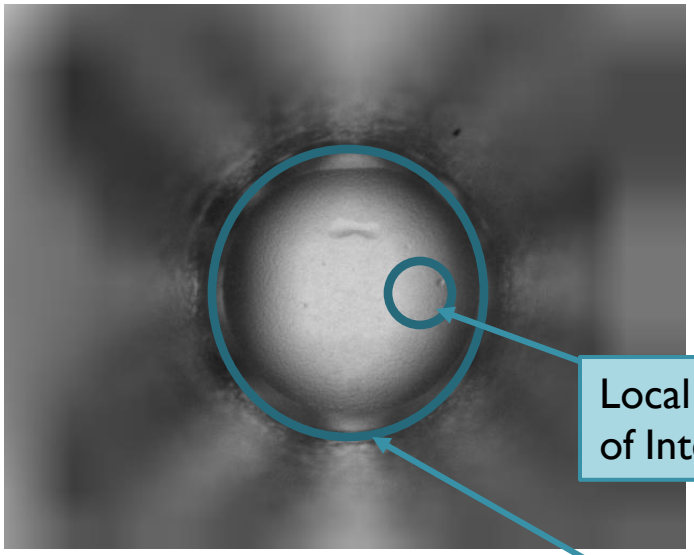
Help Conquer Cancer: Using GPUs to Accelerate Protein Crystallography Image Analysis

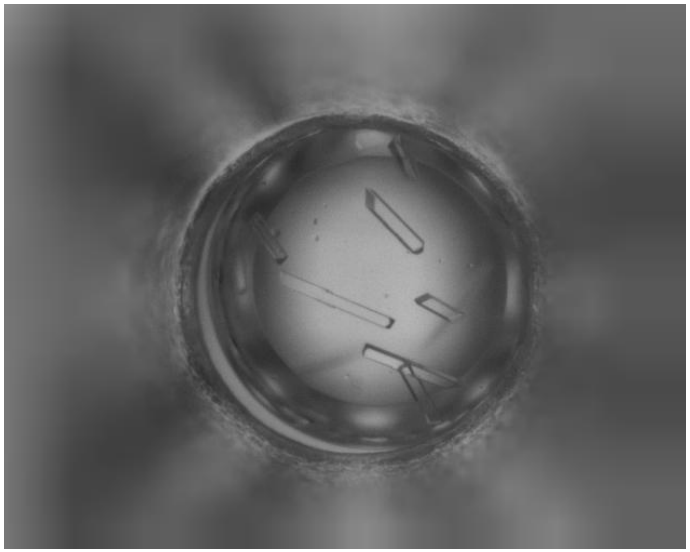
Roy Bryant, Adin Scannell, Olga Irzak, Christian A. Cumbaa

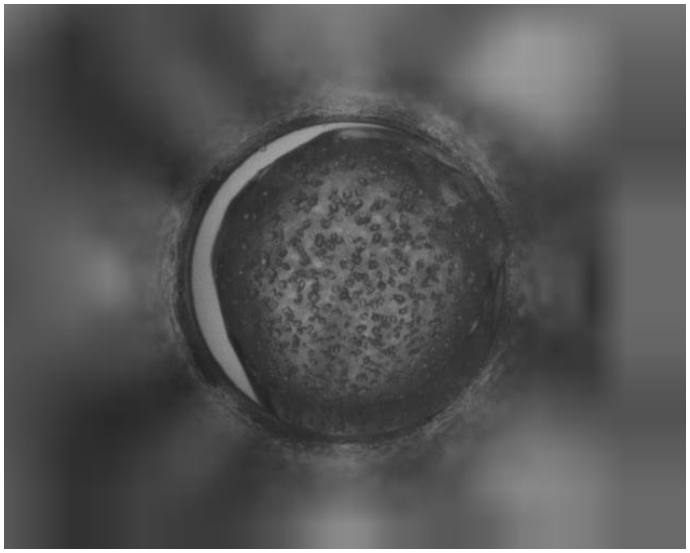
Help Conquer Cancer project

- X-ray crystallography reveals protein structure
- crystallizing the protein is difficult
  - Many thousands of experiments. Few form Crystals.
  - Automatically filter images with image feature extraction and machine learning
- over 100 million images to process
  - world community grid (250,000 PCs)
  - Will finish in 2015
- Our project: speeding up image processing

Sample Images







Sequential Code

- Approx 2 hour run time on very fast PC
- Generate GLCMs
  - grey level co-occurrence matrices
  - one for each region of interest (16 pix radius around every pixel)
  - 66 million per image takes 40% of execution time
  - **Highly optimized** - GLCMs generated incrementally
- Extract features
  - 60% of execution time
  - called 66 million times

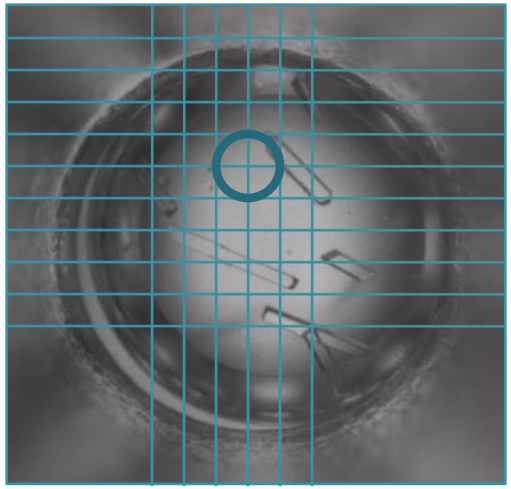
Naïve GPU Approach - Impractical

- Parallelize feature extraction
  - kernel would be called 66 million times
  - Too much data to copy back and forth

- Build on existing histogram CUDA code
  - each thread stores it's own histogram, then merges results
  - works for 64 values, but we need 4K values

Refactoring for the GPU

- Build GLCM and extract features in integrated kernel
  - Minimize data copy



- 2D grid of blocks
  - 22k blocks
  - one per pixel = one per GLCM
  - 64 threads per block
- Kernel called 3K times
  - every angle, distance, grey level depth
- Aggregate statistics differently – keep around a lot of intermediate state

Building the GLCM

- Build histogram from 32 x 32 pixel image
- Image stored in global memory
  - threads iterate column-wise to coalesce reads
- Store GLCM in shared memory
  - Initialize column-wise to minimize bank conflicts
  - Use atomic operations for histogram
    - works only on 32bit ints, so cast 2 16-bit integers into 1 32bit and incremented by adding 1 or 2^{16}
- Masks stored in constant memory

Extracting Features

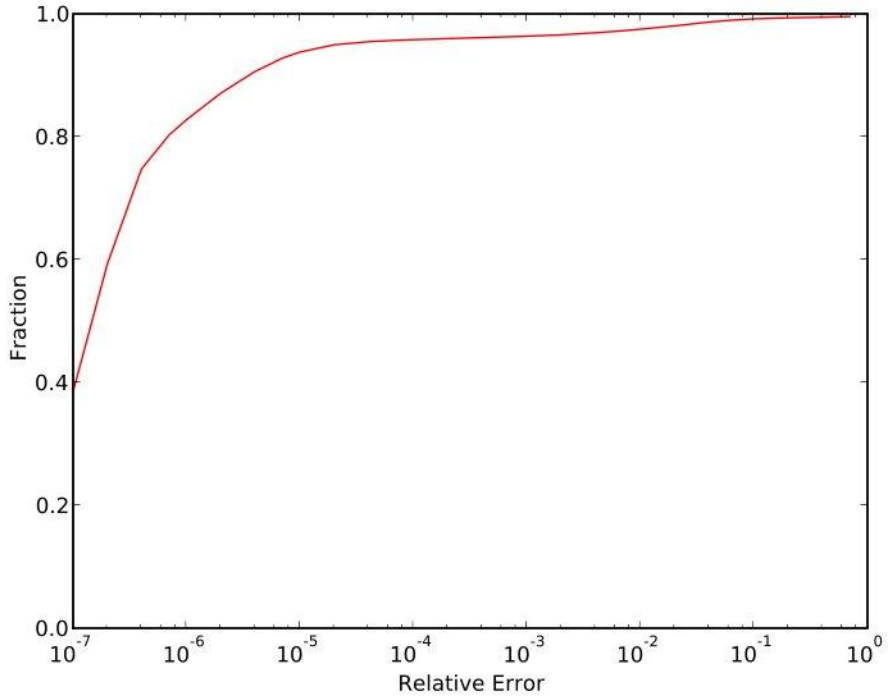
- Often sums over rows or columns
  - Iterate column-wise to avoid bank conflicts
  - Exploit matrix symmetry to change row to column iterations
- Used templates to optimize feature extraction code
  - Scaled shared memory arrays to match size of GLCM
  - Wrote tuned, unrolled summation code for each size
- Most calculation on normalized GLCM
  - Normalize on the fly since no room to store
  - Pull normalization outside loops where possible

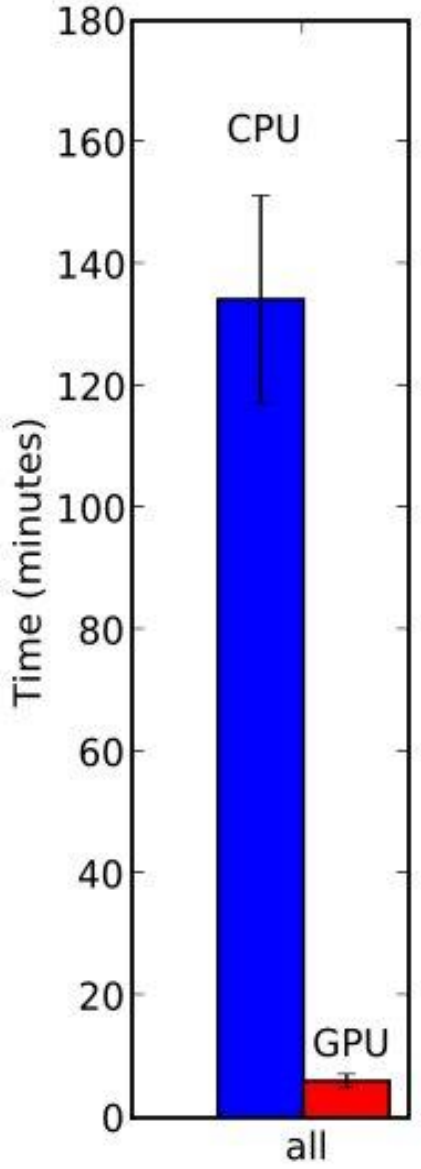
Evaluation

- Test data set included
  - With / without crystals
  - With / without precipitate
- Compared to gold standard
  - GLCM generation
  - Calculated values of features
  - Statistical summary of features

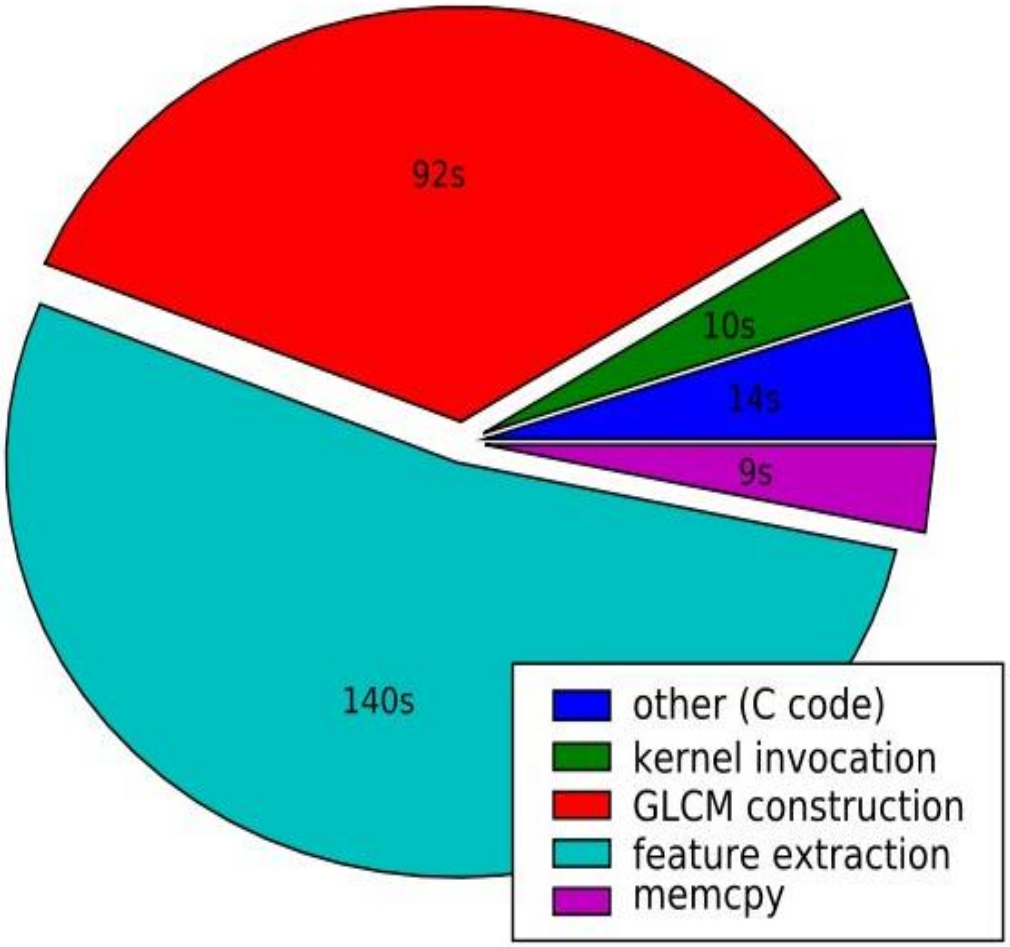
Results

- 20x execution speedup
  - 2 hours reduced to 6 minutes
- Still accurate





Runtime Breakdown



Future Steps

- Most features accurate to 5 nines
  - sqrt() and log() inaccurate for small values
  - still investigating if sufficient
  - May need to implement accurate primitives
- Further testing on variety of CUDA hardware
- HCC plans to deploy to World Community Grid