Accelerating Microscope Data Analysis Using Parallel Computing

John Ravi, Caroline Laplante (advisor), Michela Becchi (advisor)
{jjravi, claplan, mbecchi}@ncsu.edu

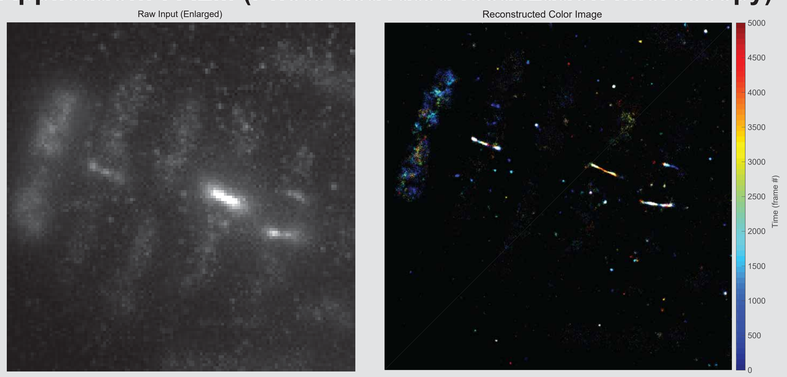
NC STATE

Abstract

Single-molecule Localization Microscopy (SMLM) techniques deal with the diffraction limit of fluorescent microscopy by localizing single molecules with high precision by stochastically switching molecules on and off. Thousands of camera frames containing subsets of blinking molecules are recorded to obtain a single super-resolution image. Each blinking molecule in each frame is subjected to localization protocols that fit the shape of the blinks, assess the quality of the blink and then estimate their center. The algorithm, implemented origininally in MATLAB and CUDA C, to compute a 'Super Resolution' image is computationally demanding. The algorithm was ported to C++ and used OpenMP to compute multiple frames in parallel.

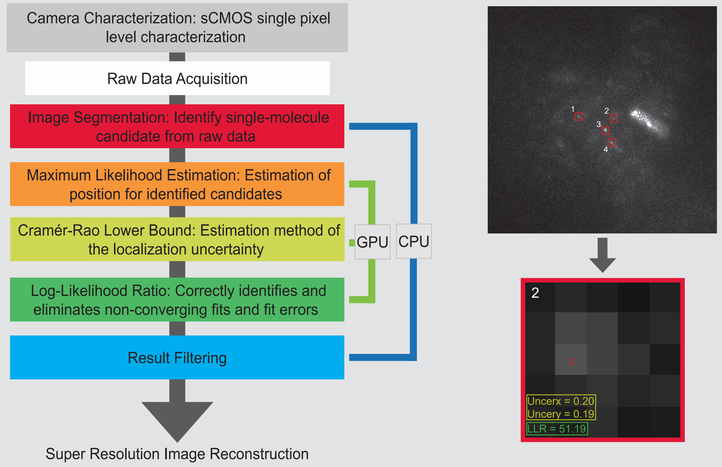
Introduction

Application: PALM (Photo-activated localization microscopy)



- Increase in resolution past the diffraction limit
- Single-molecule Localization (fitting) is computationally expensive
- Dataset above is composed of 5,000 frames with 60,468 localization
  - Raw microscope data projected over time (left)
  - Super Resolution image using color to show time (right)
    - Took 56.03 seconds to compute using existing solution

Overview



Background

Multicore CPU

- Enables simultaneous processing of multiple tasks
  - Higher performance at lower energy
- Programming Model: OpenMP
  - Shared-memory model
  - Uses compiler directives
  - Code snippet for vector addition:

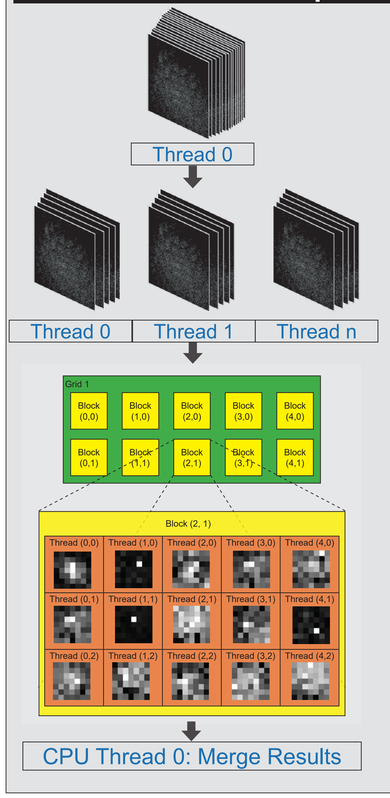
```
#pragma omp parallel shared(a,b,c) private(i)
{
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];
}
```

GPU (Graphics Processing Unit)

- Architecture designed for massively parallel algorithms
  - More cores, slower clock speeds, less complex
  - Image processing, Machine Learning, etc.
- Programming Model: CUDA
  - API created by Nvidia for GPGPU (General-Purpose computing on GPUs)
  - Code snippet for vector addition:

```
__global__ void vecAdd(double *a, double *b, double *c, int n)
{
  int id = blockIdx.x*blockDim.x+threadIdx.x;  // Get global thread ID
  if (id < n) // Make sure to not go out of bounds
    c[id] = a[id] + b[id];
}
```

Implementation



CPU Thread 0

- Input: dataset to analyze
- Split workload between available CPU threads
- OpenMP for loop directive
  - Subset of frames map to a thread

CPU-level Parallelism

- Account for sCMOS camera noise
- Identify likely candidates for localization
- Generate a list of subregions (7x7 pixel values) to send to GPU
- Use CUDA Streams to copy data to GPU

GPU-level Parallelism

- Custom CUDA C kernel
  - Fit the centers of subregions
  - Calculate uncertainty for each localization
- Each CPU thread invokes a separate CUDA kernel
  - Enables utilization of Multiple GPUs

CPU-level Parallelism

- Filter less accurate localizations

CPU Thread 0

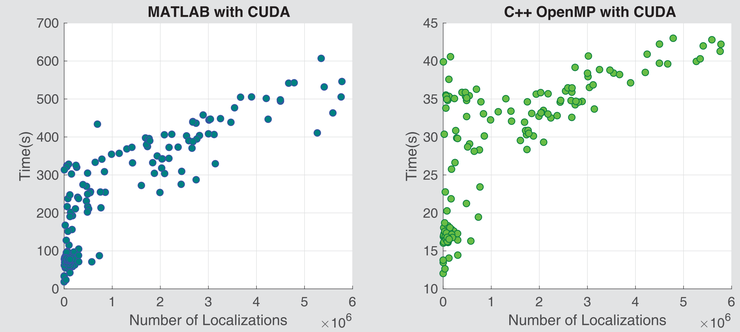
- Output: Reconstruct Super Resolution image using color to differentiate time
- Output: Save localizations to MATLAB data file for quantitative analysis

Results

Experimental setup

Attributes	Values
Type	8-core Intel Xeon(R) CPU E5-2620 v4 @ 2.10 GHz, 20M Cache
Storage	110 GB RAM, 256 GB SSD, 4TB HDD
GPUs	2x Nvidia Titan Xp (Pascal, 3840 CUDA cores, 12 GB GDDR5X)
Software	Ubuntu 16.04, g++ 5.4, CUDA 9.0, MATLAB R2016b

Results



- Plots above show the time to compute a super resolution image
  - Analyzed 161 datasets
  - Datasets varied in the number of frames (1,000 to 20,000 frames)
  - Dense samples are more computationally demanding
- On Average 8x speedup from the MATLAB implementation
  - Minimum 1.5x speedup
    - Number of Localizations: 7
  - Maximum 15.1x speedup
    - Number of Localizations: 5,346,338

Impact

- Optimize data collection parameters while acquiring subsequent datasets
- Closer to realtime SMLM imaging and analysis
- Analyze larger datasets on the same hardware

Conclusion

Future Work

- Implement another level of parallelism using OpenMPI for large clusters
- Offload more computation to the GPU
  - Gaussian Filter (2D Convolution)
  - Dilation (Morphology)
  - Results Filtering and Image Reconstruction

Acknowledgements

- MATLAB implementation provided by Dr. Huang Fang and Dr. Caroline Laplante
- Funding from the NSF Research Experiences for Undergraduates program and the NCSU ECE department

References

[1] C. Laplante et al, "High-speed super-resolution imaging of live fission yeast cells," in Yeast Cytokinesis: Methods and Protocols, A. Sanchez-Diaz and P. Perez, Eds. 2016, Available: https://doi.org/10.1007/978-1-4939-3145-3_4. DOI: 10.1007/978-1-4939-3145-3_4.