Accelerating Microscope Data Analysis Using Parallel Computing

John Ravi
North Carolina State University
Raleigh, NC, USA
jjravi@ncsu.edu

Advisor: Carolina Laplante
North Carolina State University
Raleigh, NC, USA
clplante@ncsu.edu

Advisor: Michela Becchi
North Carolina State University
Raleigh, NC, USA
mbecchi@ncsu.edu

1 INTRODUCTION

Fluorescence photoactivation localization microscopy, or FPALM, is a microscope imaging technique that generates a super resolution image (resolution beyond the diffraction limit) by using a sequence of low resolution microscope images. However, the computing process is computationally expensive due to that every light source in each image in a dataset needs to be analyzed. Moreover, a single dataset comprises 8,000 or 16,000 images of 256x256 pixels, and usually multiple datasets are acquired in a sample. To reduce interruptions between the acquisitions of dataset, the datasets are first captured and then analyzed overnight. Accelerating the analysis of the data allows for better parameter tuning of subsequent acquisitions and enables the analysis of larger datasets.

The aim of this project was to accelerate an existing FPALM implementation with major parts written in MATLAB and some computational kernels accelerated on GPU. After profiling the code, we identified two sources of inefficiency (opportunities for improvement). First, the portion that implemented in MATLAB was time consuming and could be significantly improved if ported to C or C++. Second, frames are processed sequentially, and significant speedup could be achieved by leveraging frame-level parallelism. By overcoming these bottlenecks, we achieve a 15x speedup when processing 16,000 images on an 8-core Intel server equipped with 2 Nvidia Titan Xp GPUs.

KEYWORDS

FPALM, super resolution, OpenMP, CUDA

2 METHODS

Figure 1 illustrates the FPALM pipeline. **Camera Characterization** of the sCMOS camera noise involves capturing a series of dark images with no light source to characterize each pixel offset and variance, and a series of images over a range of light intensity levels to characterize each pixel gain. **Image Segmentation** filters the images and isolates square subregions of potential single molecule emitters. **Maximum Likelihood Estimation** fits each subregion to a 2D gaussian, while the uncertainties and goodness of fit metric are calculated using the **Cramer-Rao Lower Bound** and **Log-Likelihood Ratio**, respectively. The fitting of localizations and confidence steps are done in one combined CUDA kernel on the GPU, while the subregion identification and result filtering are done on the CPU. We ported the implementation entirely to C++, kept the existing CUDA kernel for GPU parallelism, and exposed inter-frame parallelism using OpenMP.

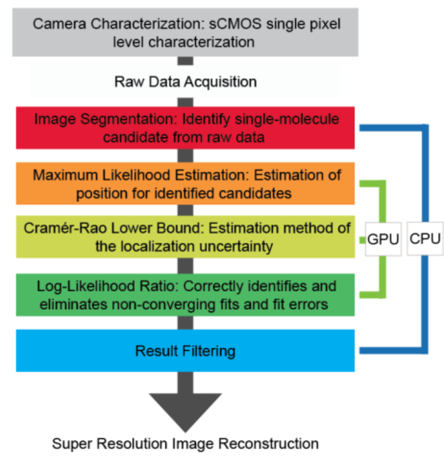

Figure 1: FPALM Algorithm Steps

We leverage frame-level parallelism to optimize performance for a single node computer. Specifically, we parallelized the code by distributing the frames in the dataset equally to all available CPU threads (Thread 0 scheduled the workload). The CPU code handles two steps Image Segmentation and Result Filter steps. For Image Segmentation, we perform a separable 2D convolution operation and dilation to filter out certain noise and identify bright-lit subregions. We avoid using GPU in this step because otherwise it would involve copying the entire dataset to GPU memory and invoke significant CPU-GPU memory transfer. Performing this step on CPU only requires a list of 7x7 pixel subregions to be sent to GPU and the statistics of each subregion (e.g., the position of the center of photon emission and confidence) to be retrieved from GPU for later CPU steps such as result filtering and super resolution image reconstruction.

Since CPU threads execute in parallel and they invoke CUDA kernel independently, our frame-level parallelism will maximize the kernel concurrency on GPU, leading to better throughput. The Nvidia Titan Xp allows for 32 concurrent kernels. To have as many concurren kernels on GPU as possible, each kernel should be properly configured. Thus, we tuned the batch size of frames that each CPU thread computes at once. This method also overlapped data transfers with kernel execution to hide memory transfer time. If multiple GPUs are present in the system, workloads are distributed to available GPUs equally.

3 RESULTS

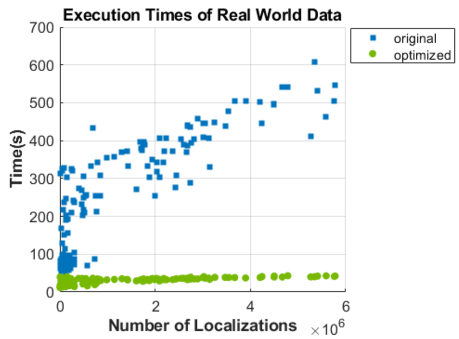

Figure 2: Execution times based on photon emission density.

Figure 2 reports the results with an Intel Xeon CPU E5-2620 v4 and 2 Nvidia Titan Xps. We use g++ 5.4.0 and CUDA 9.0. Each point on the graph represents a dataset containing either 8,000 or 16,000 images with varying photon emission density. We show the execution time of the original implementation in blue and the optimized parallel CPU and GPU implementation in green. The number of photon emissions was chosen to represent computational complexity instead of the number of frames because a dataset of 8,000 frames can be more computationally heavy if the number of photon emissions is greater than that of a 16,000 frames dataset. This occurs if the laser power is too intense, forcing the photon emitting molecules to photo bleach quickly. As can be seen, by adding CPU level parallelism, the amount of computational time required decreased by ~15x for the densest dataset analyzed.

4 CONCLUSION

This speedup in microscope data analysis allowed for instant visualization of the data at collecting time. Future work includes enabling node-level parallelism using OpenMPI. However, the data transfer between systems might be a bottleneck. This work showed that the CPU parallelism along with GPU parallelism yields the highest performance gain.

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.