Affine Transformations in OQM using a GPU

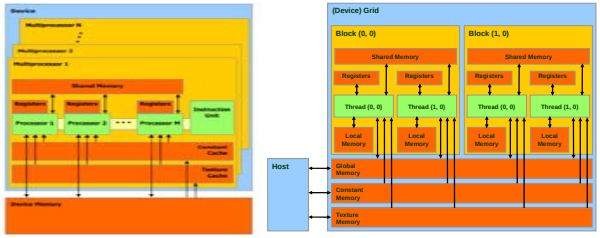
Perhaad Mistry; David Kaeli, Ph.D

Department of Electrical and Computer Engineering, Northeastern University
{pmistry,kaeli}@ece.neu.edu

Abstract

- Optical Quadrature Microscopy (OQM) is a technique that is used for imaging of an object by producing interferograms of the sample using different beams of light each having a different phase.
- Affine Transformations carried out on acquired images to adjust orientation of the images.
- Dark signal subtraction done to remove the effects of imperfections in the grabbing mechanism and detector voltages.
- Affine transforms are data parallel operations which can take advantage of massively parallel hardware.
- NVIDIA's **Compute Unified Device Architecture (CUDA)** allows programmers to harness GPU hardware using simple extensions to C.

GPU (NVidia GeForce 8800 GTX) and CUDA



(a) Multiprocessors Architecture (b) Thread and Memory Model

Figure 1. Architecture of NVIDIA GeForce 80 Series [1]

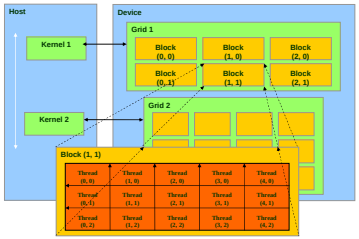


Figure 2. CUDA Thread Batching [1]

Affine Transformation

- Affine Transforms needed in OQM since imaging is done on basis of the phase difference between pixels.
- Used to adjust the grabbed images such that position of each pixel is the same within all 4 images.
- Equations of the Affine Transform are given below,

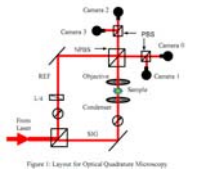


Figure 1: Layout for Optical Quadrature Microscopy

Camera 0: $|E_{ref}|^2 + |E_{sig}|^2 + 2\mathrm{Re}(E_{ref} \cdot E_{sig}^*)$
Camera 1: $|E_{ref}|^2 + |E_{sig}|^2 + 2\mathrm{Im}(E_{ref} \cdot E_{sig}^*)$
Camera 2: $|E_{ref}|^2 + |E_{sig}|^2 - 2\mathrm{Re}(E_{ref} \cdot E_{sig}^*)$
Camera 3: $|E_{ref}|^2 + |E_{sig}|^2 - 2\mathrm{Im}(E_{ref} \cdot E_{sig}^*)$.

$$M_0 = |E_{ref}|^2 + |E_{sig}|^2 - 2\mathrm{Re}(i^n \cdot E_{ref} \cdot E_{sig}^*)$$

Consider a pixel at (x,y). If it is rotated by Θ
The new pixel position is given by (x_1, y_1)
Rotation Matrix

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Shearing Matrices, if the shearing is defined by sh(x) or sh(y)

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 1 & sh(y) \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 1 & sh(y) \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Combining rotation & shearing along with matrix scaling and displacement we get

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} s(x)*\cos(\theta) & sh(y)*\sin(\theta) \\ -sh(x)*\sin(\theta) & s(y)*\cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a \\ b \end{pmatrix}$$

Affine Transformation on CUDA

- Each thread works on one Pixel,
- Groups of threads (working on groups of pixels) are separated into blocks. A block is executed on one multiprocessor on the GPU.

Algorithm for Each Pixel located at (x,y) – Repeated for all threads

- Load Affine Matrix f()
- Calculate location of pixel (x,y) (By looking at block & Thread id)
- New Position (x1,y1) = (f(x),f(y))
- Store Pixel value in (x,y) in (f(x),f(y))

Last step causes memory bottlenecks since accesses are not aligned.

Noise Subtraction on CUDA

- Fixed pattern levels not removed by common mode rejection (beam splitters) or dark subtraction since they are different for each camera.
- Requires the subtraction of Sn and Rn from Mixed Image.
- Implemented on GPU using **CUBLAS** (BLAS Implementation for CUDA)

$$E = \frac{1}{4}\sum_{n=0}^{n=3} i^n * \frac{M_n - S_n - R_n}{\sqrt{R_n}}$$

- Angle of this complex number is the phase information for the sample.

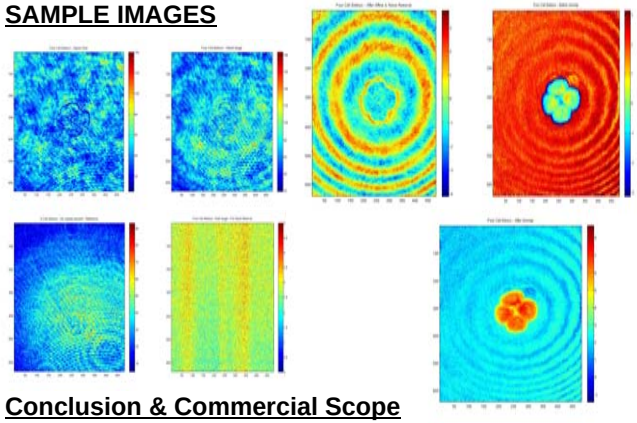
Performance Improvements

- Serial Version in MATLAB: **1.5 seconds**
- CUDA Version: NVIDIA G8800 GTX = **0.2seconds ~ 8X Speedup for heavily memory bound workload.**
- Improvement in GPU based phase unwrap code due to removal of disk activity **= 2.8sec to 2sec ~ 30% less time**

Matlab External Interface (MEX)

- GPU code needs to be accessible from within MATLAB.
- MEX files are dynamically linked libraries produced by MATLAB, allowing us to call C code within MATLAB.
- Argument passing between MATLAB and C uses the mxArray which is a C structure used by MATLAB to store data is made accessible to our code using pointer accesses.
- Using MEX wrappers we combine frame-grabbing, Affine Transforms, Noise Removal and Phase Unwrap into one automated system.

SAMPLE IMAGES



Conclusion & Commercial Scope

- Affine Transform algorithm has been implemented on the GPU.
- Using MEX has helped to remove unnecessary disk IO from the critical path and regular use case.
- MEX yields performance improvements not only in the Affine Transform but also in the phase unwrap part of this project as well.
- Optimizing memory access, availability of asynchronous memory transfer and processing on the GPU would yield further improvements.
- Thus by implementing Affine Transforms and Noise removal on CUDA and by integrating the different components of our system, we get closer to the Optical Science Lab's goal of real time OQM imaging.

References

[1] NVIDIA, CUDA Programming Guide, http://www.nvidia.com/object/cuda_home.html

[2] Combining Optical Quadrature and Differential Interference Contrast to facilitate embryonic cell counting with Fluorescence Imaging for confirmation William C. Warger II,Judith A. Newmark, ChihChing Chang, Dana H. Brooks, Carol M. Warner, Charles A. DiMarzio

[3] Master's Thesis – Sherman Braganza

[4] NVIDIA CUBLAS Manual

Acknowledgement

This work was supported in part by Gordon-CenSSIS, The Bernard M. Gordon Center for Subsurface Sensing and Imaging Systems, under the Engineering Research Centers Program of the National Science Foundation (Award Number EEC-9986821).