6.375 Proposal: Adaptive PIV

Robin Deits
rdeits@csail.mit.edu

April 7, 2013

1 Background: PIV

Particle Image Velocimetry (PIV) is an optical approach to measuring the flow field of a fluid, and has been used in the study of combustion, water flow, robotics, and many other fields. It involves seeding a fluid with tracking particles and using a laser or other planar lighting system to capture sequential images of the particle positions in a single thin 2D slice of the fluid. By comparing the change in position of groups of particles between the subsequent frames, a measurement of the local flow vector can be computed for each region of the fluid. This process of determining the movement of each section of the image is extremely time-consuming in a sequential programming system, but can be readily parallelized to significantly improve performance [2]

Each PIV computation is performed on a pair of sequential images. Computation of the fluid flow begins by dividing the image up into small windows of, for example, 64px on a side. A small window size helps ensure that all of the particles within the window move with the same velocity between the two frames. For each window, we extract the subimage corresponding to that window from the first image in the pair. We will call this subimage A. We then extract a set of subimages $B_{\Delta x, \Delta y}$ by shifting the original window in two dimensions and extracting the corresponding subimages from the second image in the pair. We can then perform a cross-correlation between A and each $B_{i,j}$ and determine the shift in x and y which maximizes the correlation. This gives the most likely location of the particles from window A in the second frame, and thus indicates the movement of that section of the fluid between the frames.

2 Adaptive PIV

Standard PIV algorithms involve an even spatial distribution of interrogation windows A with a fixed window size and some fixed overlap, such as 64 px windows beginning every 16 px. However, in order to achieve sufficient accuracy in busy fluid flows, it can be necessary to choose very small windows or very high degrees of overlap, which increases the computational demands by requiring far more cross-correlation computations. Theunissen et al. proposed a method for improving the performance of PIV in sub-optimial conditions, called Adaptive PIV [1]. Their method uses information about the current density of seeding particles and the prior estimate of the velocity field to update the size and spatial frequency of the interrogation windows A. This has the effect of increasing the number of data points in the busiest (highest particle density and highest velocity) parts of the fluid and reducing the number of samples in the most stable areas of the fluid, which can improve the amount of relevant data collected per computational unit.

In this project, I will focus on implementing Adaptive PIV on an FPGA to improve computational performance, with the ultimate goal of allowing accurate real-time fluid tracking. I will be expanding on prior work implementing a standard PIV algorithm on an FPGA [2]. I will also be using a recent MATLAB implementation of the Adaptive PIV algorithm by Samvaran Sharma of the Robot Locomotion Group at MIT CSAIL as the reference code for my implementation.

The primary benefit of this project should be the parallelization and speedup of the Adaptive PIV algorithm. In order to achieve the desired image size and accuracy, Sharma's current software requires approximately 2.5 seconds per pair of frames, which makes real-time analysis of the fluid flow impossible. In contrast, Yu et al. were able to compute 15 image pairs per second using their FPGA implementation. My goal will be to achieve this result with the added benefits of the adaptive algorithm's focus on the most important areas of the fluid flow.

3 Implementation

I will divide the implementation of the PIV system up into the high-level logic, which will be performed in MATLAB, and the computationally intensive and parallelizable cross-correlation which will be performed on the FPGA. This division is shown in Figure 1. The host machine will read image pairs from disk (simulating live capture from a camera system), then convert them to 8-bit grayscale. These image pairs will be transmitted over SCEMI to the FPGA, which will store both images in DRAM. The host machine will then select a series of interrogation windows. The exact method of selection will depend on whether we are performing normal or adaptive PIV. The host will transmit the coordinates of the interrogation windows to the DUT. The FPGA will then extract the actual image data for each window, perform the cross-correlation, and locate the peak in the cross-correlation value signifying the displacement of the particles in the window. The FPGA will then send the displacement back to the host.

3.1 Module: Window Tracker

The FPGA will contain many instances of the Window Tracker module, each of which will perform the cross-correlation and displacement extraction for a single interrogation window. The Window Tracker will be implemented as a server with the following operations:

1. put(x, y): begin calculating the cross-correlation and displacement for a new window located at (x, y) in the image
2. u, v = get(): get the displacement of the interrogation window which was previously input to put()

The Window Tracker will have a number of submodules, described below:

3.1.1 Module: Window Manager

The Window Manager will be responsible for extracting the correct pixel values from the images stored in the DRAM. It needs to support the following operations:

1. reset(x, y): begin extracting pixels for a new interrogation window located at the point (x, y) in the first image
2. px = getNextPixel(): return the value of the next pixel needed for cross-correlation computation. Since the order in which pixel values are accessed is fixed, this function needs no input.

The fixed access order for pixel values used in the cross-correlation algorithm means that we can easily make the Window Manager's access to the DRAM fully pipelined, and we can also cache values from the DRAM to reduce the number of memory accesses.

3.1.2 Module: Accumulator

The Accumulator will take pixel values from the Window Manager and compute a running sum of the product of its pairs of inputs. It will automatically reset its internal total to zero for each element in the cross-correlation matrix output.

1. update(px1, px2): add px1 * px2 to the internal total. This total is copied to result and then reset to zero after every c calls to update(), where c is the number of multiplications and additions required per element in the cross-correlation matrix
2. result = get(): Return result and then set result to Invalid

3.1.3 Module: Tracker

The Tracker determines from the output of the Accumulator the current displacement values with the highest cross-correlation between the windows. It polls the get() method of the Accumulator and maintains internal variables for the highest result seen so far and the x and y displacements corresponding to it. Once all cross-correlation computations are finished, it makes its result available.

1. x_disp, y_disp = get(): Return the displacement in x and y of the peak of the cross-correlation matrix

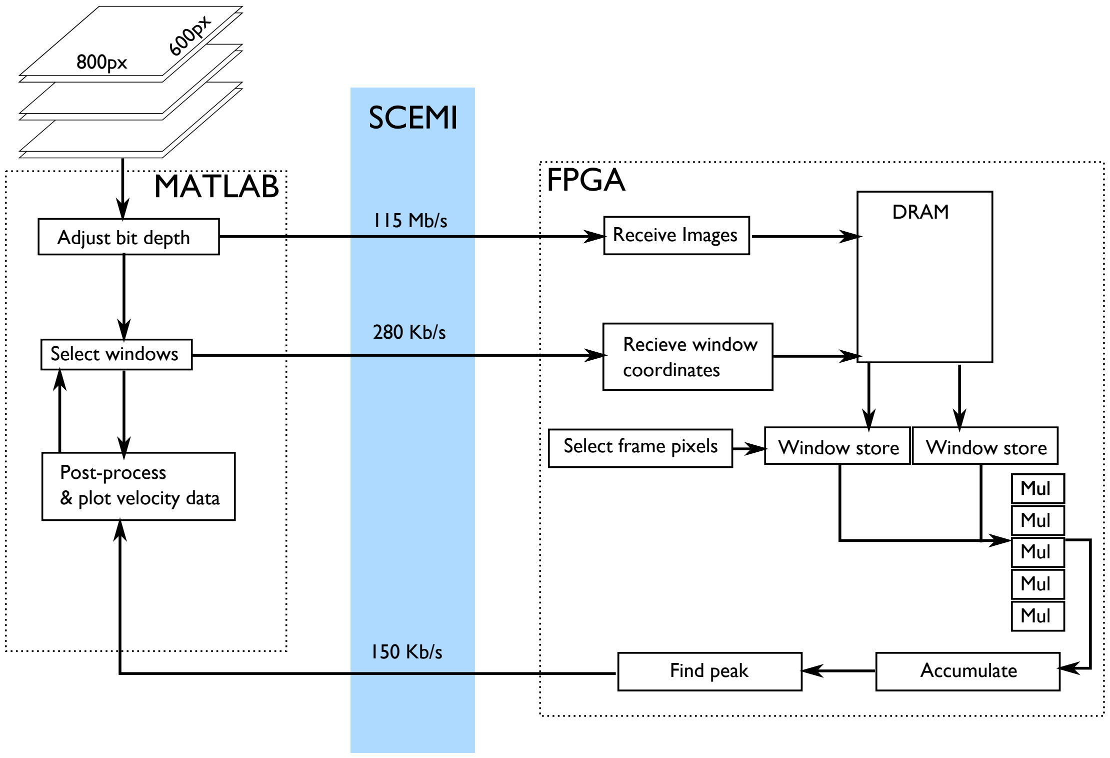

Figure 1: The system diagram for the Adaptive PIV implementation on the FPGA. High-level logic relating to the particular PIV implementation is performed in MATLAB, and the cross-correlation is performed on the FPGA.

References

[1] Raf Theunissen, Fulvio Scarano, and Michel L Riethmuller. Spatially adaptive PIV interrogation based on data ensemble. *Experiments in Fluids*, 48(5):875–887, November 2009.

[2] Haiqian Yu, Miriam Leeser, Gilead Tadmor, and Stefan Siegel. Real-time Particle Image Velocimetry for feedback loops using FPGA implementation. *Journal of Aerospace Computing, Information and Communication*, 3(2):52–62, 2006.