CATEGORY: **COMPUTER VISION - CV09**

POSTER
P4253

CONTACT NAME
Christopher Mauzey: cmauzey@mail.csuchico.edu

GPU TECHNOLOGY CONFERENCE

Wavelet-Based Optical Flow for Real-Time Wind Estimation Using CUDA

Chris Mauzey
cmauzey@mail.csuchico.edu

Pierre Derian
pderian@csuchico.edu

Shane D. Mayor
sdmayor@csuchico.edu

California State University Chico

Purpose

Above: The Raman-shifted Eye-safe Aerosol Lidar (REAL) at California State University Chico.

The REAL is an atmospheric light detection and ranging (**lidar**) system. It produces near-horizontal and vertical cross-sectional **images of the lower atmosphere**. The images reveal the spatial distribution of **atmospheric aerosol** (particulate matter). By applying motion estimation algorithms to image sequences, two-dimensional wind velocity fields can be determined.

In our previous work, we applied **cross correlation** (a technique commonly used in partical image velocimetry) to estimate the motions of aerosol features and thus estimate remote wind velocity [1]. We have applied this method to **real-time** wind velocity estimation by utilizing GPU computing to accelerate cross correlation algorithms [2]. We are now exploring the use of **wavelet-based optical flow** as another method for estimating wind velocity. Optical flow can overcome some of the limitations of the cross correlation based methods such as resolving finer levels of turbulence and vorticity. By using GPU computing, we can create a real-time wind estimation system similar to the previous one that used cross correlation.

Wavelet-based Optical Flow

Optical flow is a **computer vision** method commonly used for video compression, object tracking, and motion estimation.

Our in-house code, *Typhoon*, is dedicated to fluid motion estimation. It relies on a wavelet representation of the displacement components [3], to take advantage of the multi-scale nature of wavelet bases as well as their regularity properties. The wavelet coefficients are the unknowns to the problem. For a typical wind estimation from lidar data, it represents **0.5 to 1 million unknowns** simultaneously estimated by minimizing the **Displaced Frame Difference** model (shown below).

$$\boldsymbol{d} = \arg\min_{\boldsymbol{d}} \int_{\Omega} \left[S_2(\boldsymbol{x} + \boldsymbol{d}(\boldsymbol{x})) - S_1(\boldsymbol{x})\right]^2 d\boldsymbol{x}$$

This model minimizes the L2-norm between the first image (S_1) and the second image (S_2), which is unwarped by the displacement field (***d***).

Wavelets transforms (coefficents <=> displacement components) are the critical part of the estimation process: **each evaluation of the functional requires 2 inverse (reconstruction) and 2 forward (gradient) transforms**. Since most of the execution time of the estimations is spent calculating **discrete wavelet transforms (DWT)**, most of the development was focused on improving *Tsunami*, our in-house DWT library. *Tsunami* was designed to be generic enough to support a wide range of wavelet transform families for 1D and 2D data. It employs circular convolution decimation and expansion kernels for forward and inverse transforms, respectively. *Tsunami* can transform data with dimensions that are multiples of a power of 2 (i.e. images of size $2^k a \times 2^k b$, where k>0).

2D DWTs are calculated by performing 1D DWTs along the rows of an image, and then 1D DWTs along its columns. This makes it easy to compute in parallel on each row and on each column. This was done with the original *Tsunami* library using OpenMP `#pragma omp parallel for` on loops over rows and columns. We also apply parallel computing not only per row/column but also per element in the circular convolutions through the use of AVX SIMD instructions.

Our CUDA implementation of *Tsunami* (*CuTsunami*) expands upon the work done on CPU parallelism with the added benefits of the GPU. When performing convolutions along rows, we cache in shared memory the segment of the row that is within the filter window for all columns within a thread block. This helps mitigate the cost of reading data from global memory.

However, when applying the row-wise method to the column-wise convolutions we experienced some slow-down due to uncoalesced reads across rows (data is contiguous along rows). We apply a different approach for column-wise convolutions by having each thread in a block compute multiple elements along a row, while caching the column data in register memory. This keeps uncoalesced global reads to a mininum while recycling data in faster register memory to further reduce global reads. The code on the right employs this method.

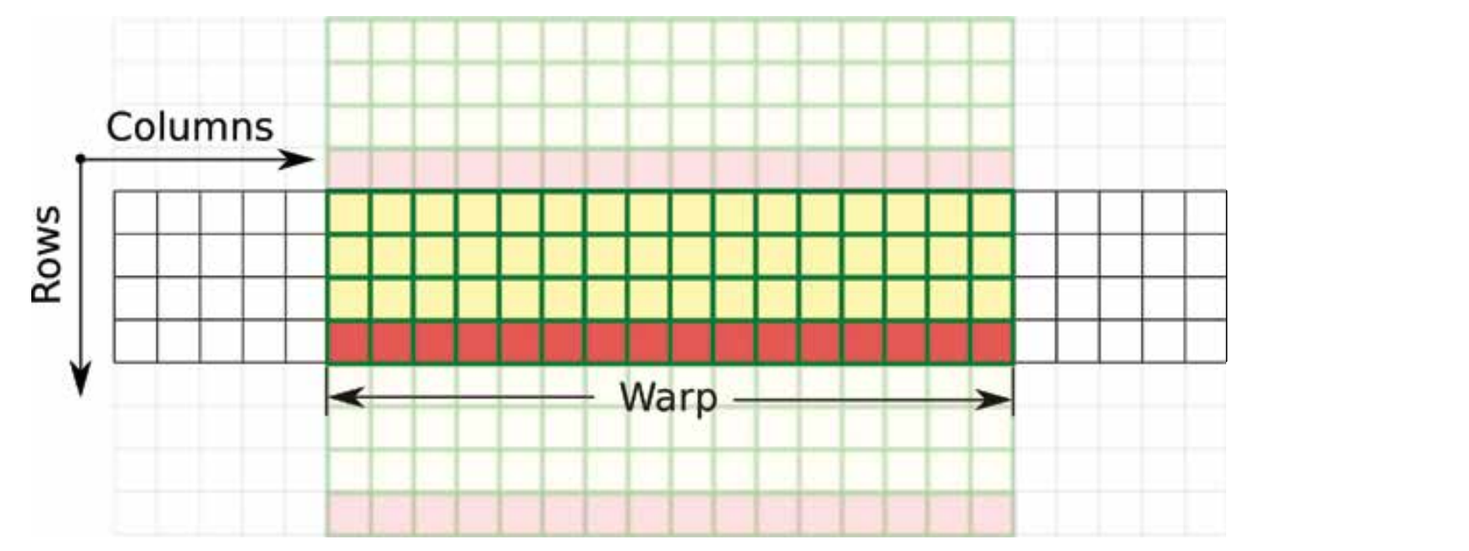

Fig. 2: Each thread in column-wise convolution calculates multiple row elements for a column. Row elements from the column are stored in a local array in register memory (yellow) while the "leading" row (red) is read from global memory. When the convolution shifts forward the row values are shifted back by one row, and the next leading row is read. This procedure is performed at every value within the filter window.

```
__device__ void decimateColumns(const double* signal, int sizeS, int nColumns,
                                const double* filter, int sizeF, int offsetF,
                                int localRowIdx, int localRowDim, double* result){
        const int sizeR = sizeS/2;
        const int sizeS_1 = sizeS-1;
        double temp[DWT_COL_ROWMULT];
        double segment[DWT_COL_ROWMULT*2];
        //-for each signal value
        for (int n = 0; n<sizeR; n+=(localRowDim*DWT_COL_ROWMULT)) {
            int rowIdx = n + (localRowIdx*DWT_COL_ROWMULT);
            if(rowIdx < sizeR){
                #pragma unroll
                for(int i=0; i<DWT_COL_ROWMULT; ++i) temp[i] = 0.0;
                int z = 2 * (rowIdx+(DWT_COL_ROWMULT-1)) - offsetF;
                z = (z<0)?((sizeS + z%sizeS)%sizeS):z%sizeS;
                #pragma unroll
                for(int i=DWT_COL_ROWMULT*2-1; i>=0; --i){
                    segment[i] = signal[nColumns*(z--)];
                    z = (z>=0)?z:sizeS_1;
                }
                //-for each filter value
                for(int k=0; k<sizeF; k++){
                    double filterVal = filter[k];
                    #pragma unroll
                    for(int i=0; i<DWT_COL_ROWMULT; ++i)
                     temp[i] += segment[2*i+1]*filterVal;
                    //shift values from the first row to the last row
                   //insert new value in first row
                    #pragma unroll
                    for(int i=DWT_COL_ROWMULT*2-1; i>0; --i)
                    segment[i] = segment[i-1];
                    segment[0] = signal[nColumns*(z--)];
                    z = (z>=0)?z:sizeS_1;
                }
                #pragma unroll
                for(int i=0; i<DWT_COL_ROWMULT; ++i)
                    if(rowIdx < sizeR-i) result[(rowIdx+i)*nColumns] = temp[i];
            }//if(rowIdx < sizeR)
        }//for(int n = 0; n<sizeR; n+=(localRowDim*DWT_COL_ROWMULT))
}
```

Fig. 3: Column-wise decimation kernel for forward wavelet transform written in CUDA C.

Concept

Scans produced by REAL are composed of radial samples of **backscatter intensity**. These values are converted to decibels and are passed through a high-pass median filter to remove large-scale features caused by backscatter attenuation [1]. The scans are then interpolated onto uniform 2D Cartesian grids, as shown in the images below. By performing motion estimation between consecutive scans, we can measure the **displacement of aerosol features** between scans. By dividing the displacement by the time difference between the scans, we can estimate the **instantaneous wind velocity** between the scans. Horizontal scans produced by REAL are usually completed **within 10 to 20 seconds**. In order to estimate wind velocity fields in real-time, we must compute motion estimations between scans within that time window.

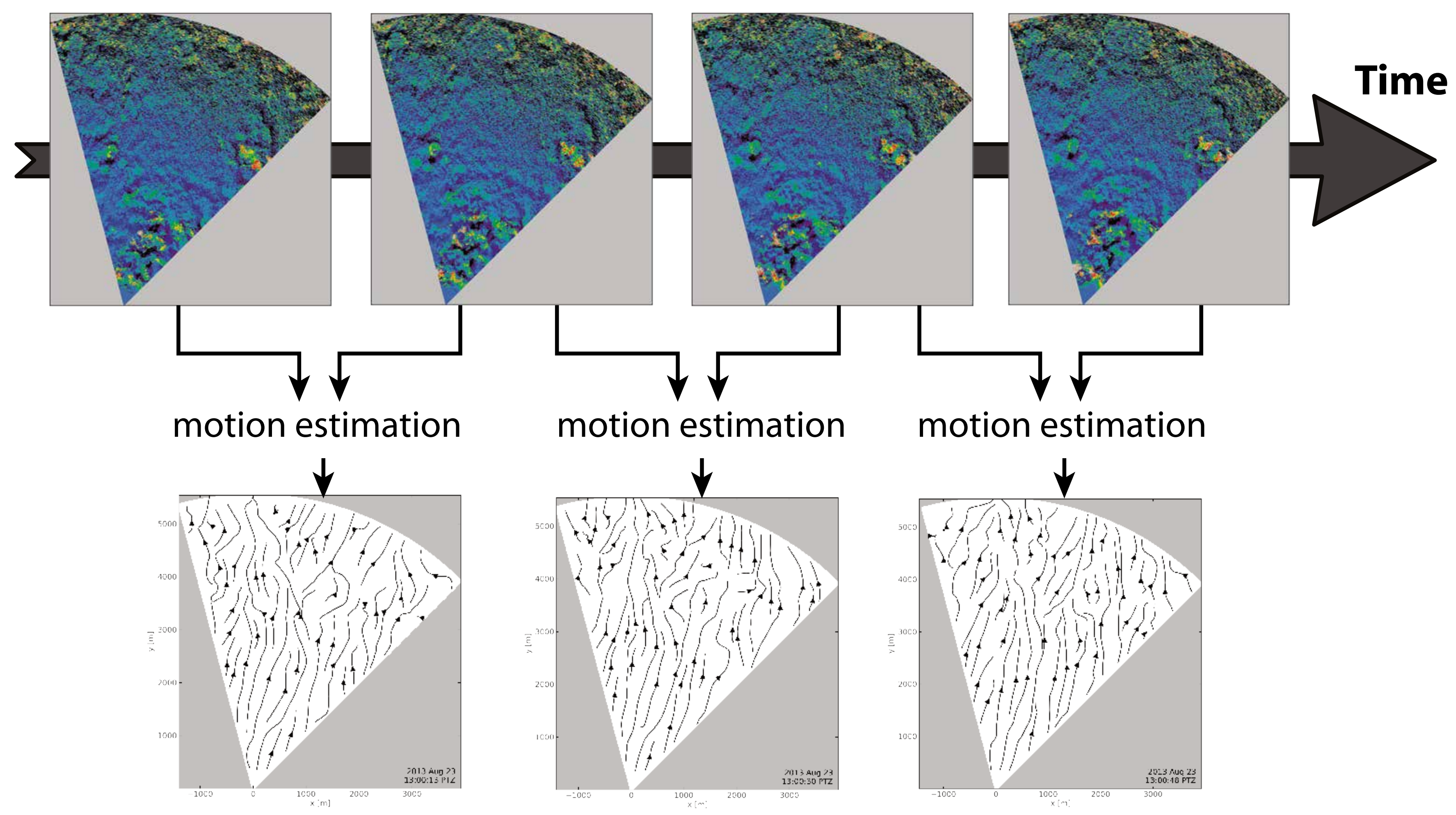

Fig.1: Scans from August 23, 2013 in Chico, CA. Pairs of consequtive scans along time (top) are passed through an optical-flow-based motion estimation program to create vector flow fields for each pair. These flow fields are rendered as streamlines (bottom).

Results

Benchmarks for the CPU and GPU versions of *Typhoon* where run on a workstation containing an **Intel Xeon E3-1225** @ 3.1 GHz, and an **NVIDIA GeForce GTX Titan**. Although multithreading and SIMD instructions in the CPU code do provide speed-up over single-threaded code, performance decreases as image size increases. The GPU code, on the other hand, becomes more efficient as image size increases. The high efficiency of our CUDA-accelerated optical flow makes it ideal for real-time wind field measurements.

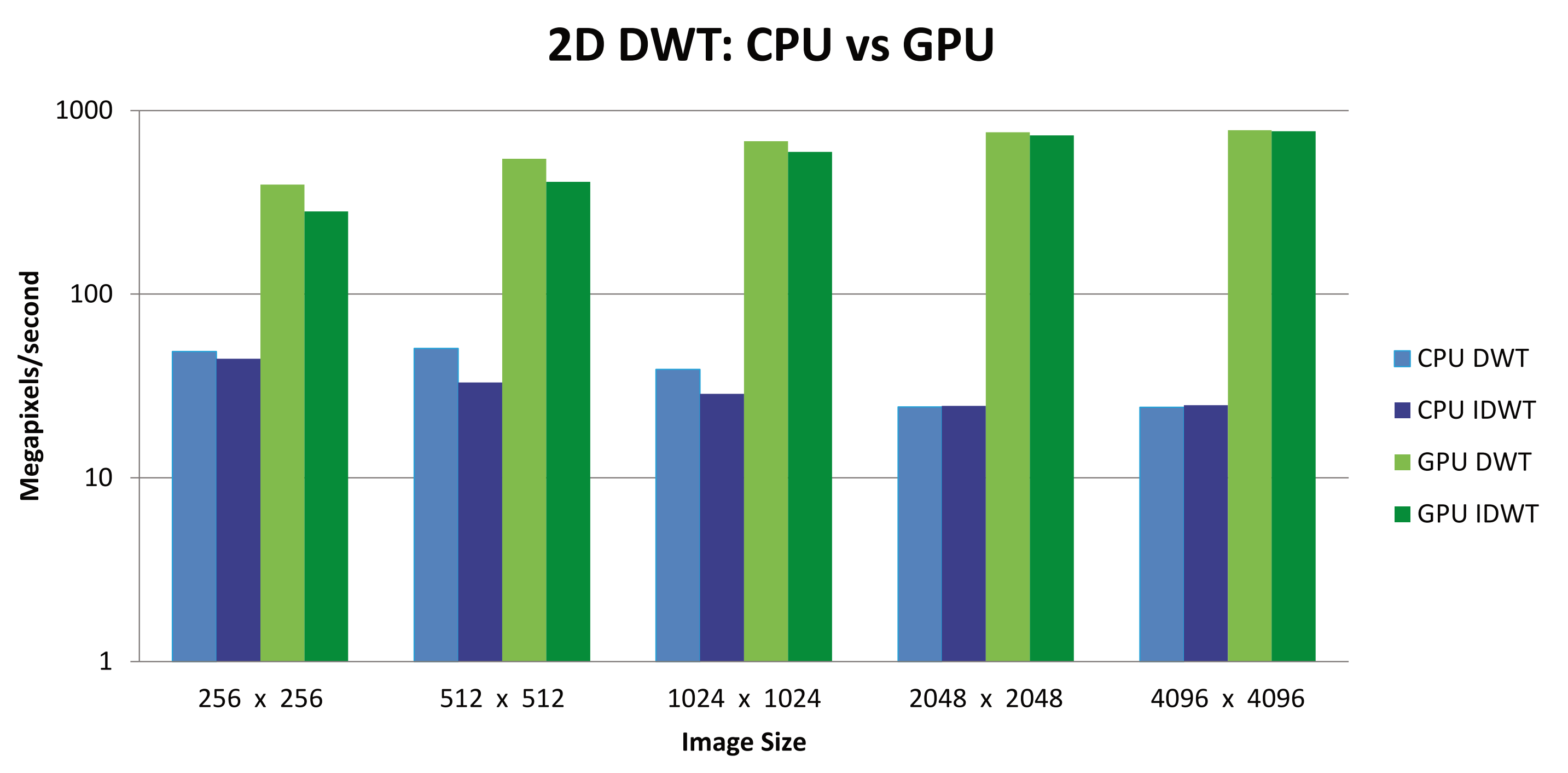

Fig. 4: *Tsunami* and *CuTsunami* benchmarked using a Daubechies wavelet with 10 vanishing moments.

Data	Image Size	CPU time (sec)	GPU time (sec)	Speed-Up
Particles	256 x 256	1.0383	0.1464	7.092
Scalar	512 x 512	6.9362	0.7266	9.546
Particles	1024 x 1024	30.968	0.9612	32.218
Satellite Water Vapor	2284 x 1339	1049.152	13.5482	77.438

Fig. 5: CPU and GPU versions of *Typhoon* benchmarked with various datasets. Runtimes were averaged over 10 runs for each set. The 256 x 256 and 1024 x 1024 images are from Particle Image Velocimetry (PIV) experiments. The 512 x 512 image is a scalar field being advected. The largest image is from an infrared water vapor satellite over North America. The efficiency of the CPU implementation degrades significantly over that of the GPU version as the image size increases.

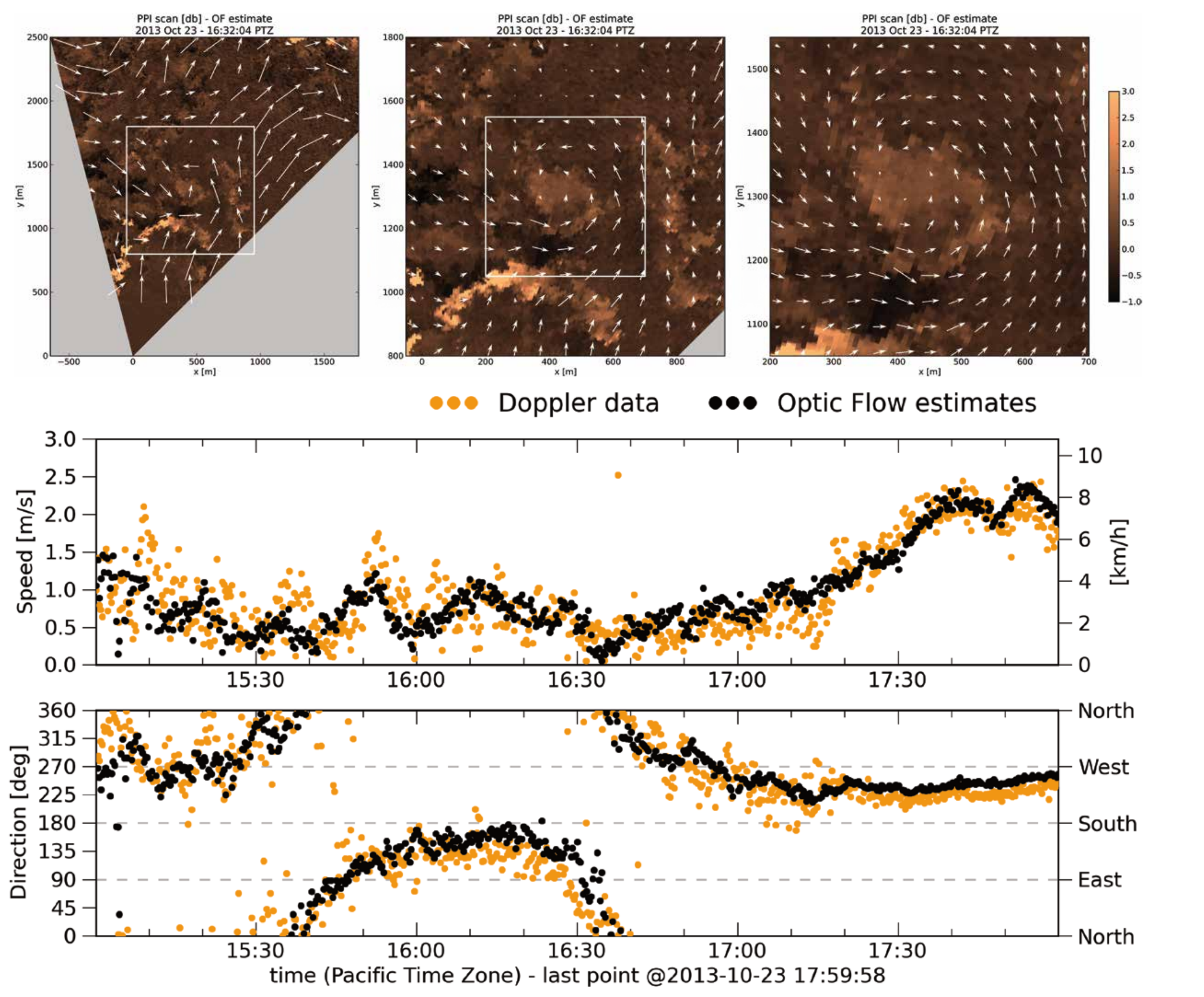

Fig. 6: Vortex case from October 23, 2013 in Chico, CA. During a turbulant day, a vortex originates from the south-west direction and heads north-east. The three images (top) show the vortex's horizontal distribution of aerosol, zooming-in closer to the vortex from left to right. Overlayed on these images are vector flow fields estimated by wavelet-based optical flow, which captured the vorticity of aerosol features. The vortex also passed over a vertical-scanning Doppler lidar that was providing in-situ wind measurements, located 1.5 km from REAL. The time series (bottom) compares the measurements of the Doppler lidar (yellow) with the velocites estimated by optical flow (black) as the vortex moves across the scan area. These measurements were taken from 100 m above ground level.

References

[1] Mayor, S. D., J. P. Lowe, and C. F. Mauzey, 2012: Two-component horizontal aerosol motion vectors in the atmospheric surface layer from a cross-correlation algorithm applied to elastic backscatter lidar data. J. Atmos. Ocean. Technol., 29, 1585-1602.
[2] Mauzey, C. F., J. P. Lowe, and S. D. Mayor, 2012: Real-time wind velocity estimation from aerosol lidar data using graphics hardware. Poster presentation AV10 at the GPU Technology Conference,14-17 May 2012, San Jose, CA.
[3] Dérian, P., Héas, P., Herzet, C., and Mémin, E. 2013: Wavelets and optical flow motion estimation. Numerical Mathematics: Methods, Theories and Applications, Vol 6, pp. 116-137.

Acknowledgements

Project funded by Grants 0924407& 1228464 from the National Science Foundation's Physical and Dynamic Meteorology Program.

Presented by the Atmospheric Lidar Research Group at California State University Chico. http://phys.csuchico.edu/lidar/