Karlsruhe Institute of Technology

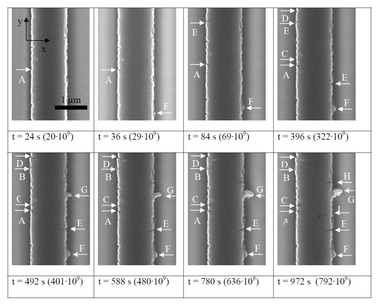
Karlsruhe Institute of Technology

GPU Computing for Real-Time Optical Measurement Techniques

Suren Chilingaryan, Sven Bundschuh, Chris Eberl, Andreas Kopmann

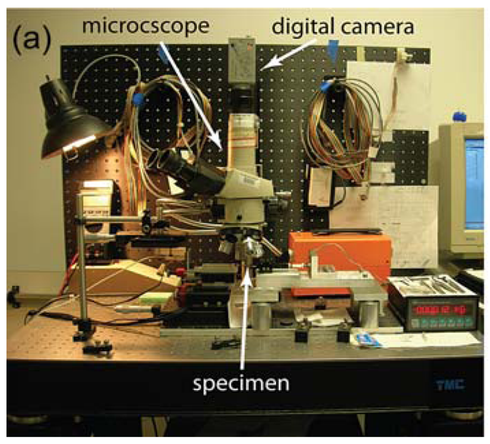
Digital Image Correlation and Tracking

The accurate measurement of displacement and strains during deformation of advanced materials and devices continues to be a primary challenge to designers and experimental mechanicians. The increasing complexity of technological devices with stringent space requirements leads to imperfect boundary conditions that have to be properly accounted for. The push toward miniaturizing devices down to nanometer length scales imparts additional difficulties in measuring strains as the application of conventional extensometers and resistance foil gages are cumbersome, damaging, or even impossible. A technique which can cover all that and also can deal with complicated strain fields in structures or structural materials is the Digital Image Correlation. With this technique, strain can be calculated from a series of consecutive images with sub pixel resolution. However, the image processing is computation intensive and using general purpose processors it is not possible to analyze images in real time. With hundreds of simple processors used to transform vertexes in 3D space modern graphic adapters offer a way to speed up the process of more than one order of magnitude at low cost and with good scalability. To use this computational power we have implemented the image tracking algorithm using CUDA.

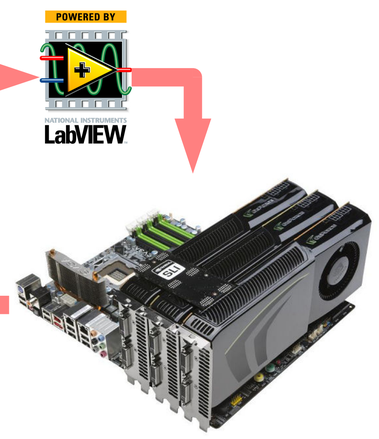


Series of micrographs showing details of the microstructural changes. The average line width is 1.1 µm.

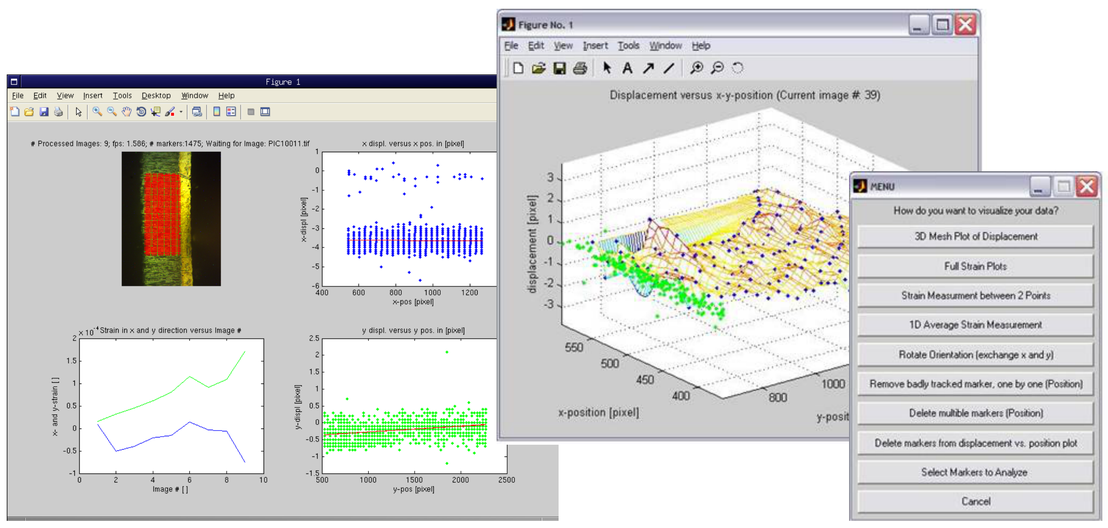
Measurement Setup



A high precision camera is mounted to the microscope with x40 magnification rate resulting in approximately 8 pixels per micron resolution.



The images are readout with a LabVIEW application and processed in real-time using NVIDIA GTX295 cards.

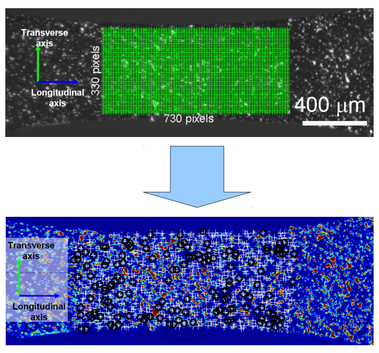


The GPU assisted MATLab application visualizes deformation of the sample in real-time and can be used to analyze the stored results later.

Algorithm & Implementation

The correlation algorithm is based on the tracking of the grey value patterns in small local neighborhood facets, typically having a size of 30x30 pixels. A set of markers (control points) is defined upon the first image. For each marker, the cross-correlation is computed between the facets of the first, template, and current images. Then, the marker is moved to the location corresponding to the maximum of the correlation. For better precision it is possible to locate the maximum correlation coefficient with the sub-pixel precision using polynomial interpolation. Monitoring the adjustments of the dense enough grid of markers, it is possible to trace formation of cracks, measure deformations and strains in materials, etc.

The original implementation uses Matlab Image Processing Toolbox. The *cpcorr* function is used to tune marker location using normalized cross-correlation. The *cpcorr* calls *normxcorr2* function to compute correlation and *findpeak* function to find it's maximum. Unfortunately, Matlab implementation is slow. Therefore, using MEX interface we have developed GPU-assisted implementation of cpcorr function. To further increase performance, our implementation supports preloading of images. While the GPUs are computing adjustments of markers, the next image is loaded and preprocessed. In addition to Matlab application, we provide a C library which can be used with 3rd party applications and a console utility.



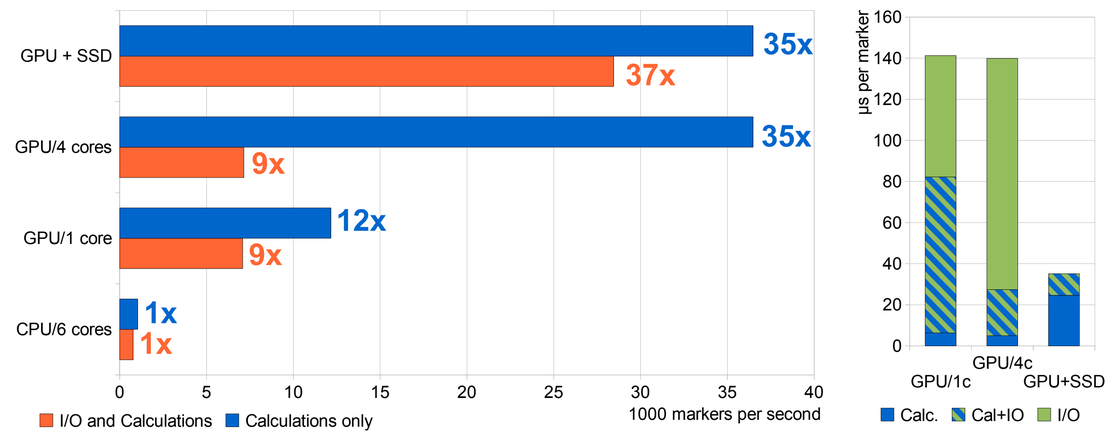
Displacements of 5000 control points in Al-Mo Thin Film measured using digital image correlation

Project Pages:
(open source)

Original version: http://www.mathworks.com/matlabcentral/fileexchange/12413
Optimized GPU version: http://dside.dyndns.org/dict

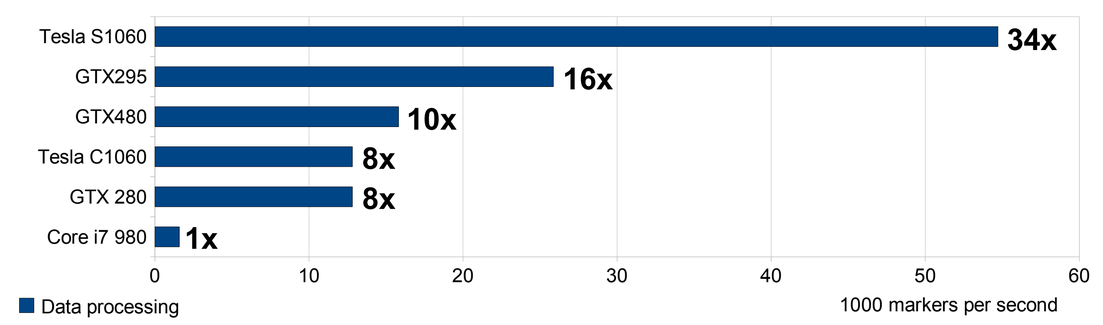
Performance Evaluation

CPU	Intel Core i7-980-X Extreme Edition (6 cores, 3.33 Ghz, 12 MB Cache)
GPU	2 x NVIDIA GeFoforce GTX 295
Motherboard	ASUS Rampage III Extreme (Intel X58, 4 PCIe x16 slots, SATA-600)
Memory	12GB DDR3 PC1333
HDD	WDC5000AACS
SSD	2 x Intel X25-E as Raid-0
Software	OpenSuSe 11.2, MATLAB 2009b, CUDA 3.0, CUDPP 1.1.1
Data	520 TIFF Images (2208x3000, 24bit), 2736 markers, 30x30 facet



The chart compares performances of original and CUDA-optimized versions using three different subsets of available resources. The computation and loading of images are performed in parallel and already using a single GPU adapter, the performance is bound by the I/O time only. Usage of fast SSD media, especially organized in Raid-0, allows to further increase performance. With 2 GTX 295, 4 cores, the GPU version is 37 times faster if compared with original CPU implementation (all 6 CPU cores are actually used by the software). The performance is GPU-bound in this case, about half of all time is spent computing FFT transforms. Therefore, installing two more cards in free PCIe slots will further increase throughput.

Evaluation of NVIDIA Hardware



The chart compares performance of NVIDIA cards available on the market according to our implementation. Only the speed of the *cpcorr* function is measured. The image loading and preprocessing as well as few other operations performed always using CPU do not contribute to the numbers presented on this chart. For fairness sake the optimized version is used to get Core i7 performance (about 30% faster compared to original version).

Optimization Steps

	Speed-up*	
Original: Original Version	**1.0**	244 / 8285 / 0 / 0
Optimized: The template image is preprocessed and cached. This eliminates half of the image reads and direct FFT transforms	**1.5**	219 / 5182 / 0 / 0
FFT plugin. NVIDIA Matlab plugin is used to compute FFT transforms	**2.0**	219 / 2745 / 731 / 528
Cross-correlations in GPU: The preprocessed data is cached in the GPU memory; results are not transfered in between of direct and inverse transforms forth and back	**4.6**	219 / 1369 / 171 / 60
Marker tracking in GPU: CUDAfication of postprocessing code: only the new coordinates of control points are transfered back to system memory	**16**	219 / 251 / 13 / 61
TIFF: The images are loaded with libtiff and converted to the greyscale using SSSE3 instructions	**34**	157 / 17 / 13 / 61
SSD: The faster storage media is used	**70**	32 / 15 / 13 / 60

0 100 200 300 400 500 600 700 800
µs per marker
GPU Time
Transfer Time
CPU Time
I/O Time

**** The time measurements was performed using older computer equipped with a single GTX280 graphic card. Image preloading was not used.***