REDUCTION OF THE DATA PROCESSING TIME FOR LOCK-IN SHEAROGRAPHY

Pierre Pfeffer¹, Daniel Hoffmann¹, Christian Kolb¹, Marcel Mayr¹, Giovanni Schober¹, Martin Bastian¹

¹SKZ-KFE gGmbH, Friedrich-Bergius-Ring 22, 97076 Würzburg, Germany
*Corresponding Author: p.pfeffer@skz.de

ABSTRACT

This paper presents possible means of reducing the computing time necessary for processing shearographic phase images obtained using a lock-in technique. The main steps to obtaining meaningful information from lock-in shearography data are filtering, unwrapping and Fourier-transformation, which can all be very time-consuming for large amounts of data. Using Matlab’s Parallel Computing Toolbox, we evaluated and compared the speed-up of processing times when using an optimized programming syntax, parallel CPU-computing or parallel GPU-computing. Coding in a time optimized syntax improved the processing time by a factor of about 10 compared with a simple syntax, employing 12 parallel computing threads leads to a time improvement factor of 29 and utilizing a GPU for computing resulted in a 116-fold speed-up. Optimized GPU-computing it thus recommended for processing large amounts of shearography data and can reduce the computation time for the data acquired by one lock-in measurement in a 5 MP resolution from about 10 hours to competitive 5 minutes.

KEYWORDS: Shearography, Image Processing, Parallelization, NDT Methods

1. INTRODUCTION

Shearography is a nondestructive testing method based on interferences of sheared speckle patterns [1]. Due to conventional shearography being a fast full field inspection technique, it has become a widely accepted method for the testing of elastomers, mainly for tires, and modern composite materials [2, 3, 4, 5]. In contrast to conventional shearography which observes the differences between an image of a test object under load and an image in an unloaded state, lock-in shearography captures multiple images during a periodic, contactless excitation of the test object. This results in lock-in shearography being able to offer higher resolutions and access to (excitation frequency dependent) depth information. The drawback, however, is the necessity to acquire much larger amounts of data which can easily mount up into the GB range and which can take several hours to process, using simple algorithms. In order to reduce the data processing time to a practical value, we compare several time optimized and parallel computing schemes.

2. METHODS

Figure 1 schematically describes the principle of a shearographic image acquisition. The test sample is illuminated by one or several expanded laser beams which create a speckle pattern on the sample’s surface. This pattern is captured by the shearographic camera, in which a beam splitter and two mirrors, one of them being tilted to produce the shearing, map the pattern on the camera sensor. Due to the shearing, the image is projected twice on the sensor, resulting in an interference pattern. By shifting one of the mirrors and thus varying one of the beam paths, a software is able to calculate phase images out of several regular amplitude images. Capturing these phase images in an unloaded and a loaded sample state and digitally subtracting those two images finally results in the difference images in which characteristic defect patterns can be observable. The example in Figure 1 for instance shows part of a plastics sample in which four mostly circular defects are discernible.
Subsequent image processing typically consists of filtering and unwrapping (i.e. compensating $2\pi$ phase jumps). The effects of these steps are demonstrated in Figure 2. Subfigure a) displays the raw difference image (comparable to Figure 1) of a polyethylene sample with 16 cylindrical millings of different depths. Subsequent image processing steps require images to be as free from noise as possible, so the difference image is usually filtered initially. This can be done using regular mean or median filters, however these tend to distort the phase values, resulting in a reduced codomain. A way to circumvent this problem is by first applying the sine and cosine functions to each pixel, afterwards filtering the results with a mean filter and finally reconstructing the filtered phase image using the arc tangent function:

\[
\begin{align*}
    s_{\text{img}} &= \sin(\text{img}) \\
    c_{\text{img}} &= \cos(\text{img}) \\
    s_{\text{img}} &= \text{conv}(s_{\text{img}}, \text{kernel}) \\
    c_{\text{img}} &= \text{conv}(c_{\text{img}}, \text{kernel}) \\
    \text{img}_{\text{filtered}} &= \text{atan}(s_{\text{img}}, c_{\text{img}}).
\end{align*}
\]

Here, img represents the original, unfiltered difference image-matrix, conv(a, b) an operation which convolves a and b, and atan(a, b) the arc tangent of a and b. Kernel designates the filter kernel in matrix form. The result of applying 10 filter iterations with a 5x5 kernel to the image of Figure 2 a) is presented in subfigure b).

Since the acquired phase values, which represent the samples’ deformations, are projected to the codomain $[0, 2\pi]$, large deformations lead to discontinuous phase jumps. After filtering, latter become clearly visible, as can be seen in subfigure 2 b), where white and black areas lie next to each other. Those discontinuities can be disposed of by subjecting the filtered image to an appropriate unwrapping function. Several algorithms differing in their mathematical approach and in their robustness to noise and image errors are known in the literature [6]. A particularly well performing algorithm based on a weighted and unweighted least-squares method is described in [7]. Since this algorithm is rather time intensive and not suited for parallelization and in order to demonstrate data processing time reduction we focus on a more basic and simple algorithm which unwraps an image by processing each of its lines or columns independently [8]. Parallelization schemes can thus be applied very easily to this algorithm, however the merely one dimensional treatment of the problem can lead to artifacts in more complex and noisy images. The unwrapped image of our example can be found in Figure 2 c). Instead of the phase jumps in subfigure b), now smooth transitions predominate and the circular pattern stemming from the millings can be observed more clearly.
Fig. 2 a) - c) Space resolved phase images of a (10x10x2) cm³ polyethylene sample with 16 cylindrical millings of different depths (inserted from the sample’s backside). a) Unprocessed difference image. b) Filtered image. c) Filtered and unwrapped image.

In contrast to conventional shearography, which only regards single difference images, lock in shearography captures a multitude of images during a continuous, periodic excitation of the sample (visualized in Figure 3). Typically this periodic excitation takes place at a single loading frequency \( f \). After the capturing, the stack of phase images is converted to a series of difference images. Since each of the resulting difference images is provided with a corresponding time stamp, a pixel-wise Fourier transformation of the image stack allows to assign the resulting amplitude or phase value at frequency \( f \) to the respective pixel and thus to obtain a result image which represents the shearographic response at the selected frequency and in which noise and perturbations occurring at different frequencies are suppressed.

Fig. 3 Schematic representation of a lock-in measurement. A series of shearographic images is taken during a continuous, periodic excitation of the test sample, difference images (cf. Figure 1) are calculated, image processing (cf. Figure 2) is performed and finally Fourier transformation and selective evaluation at a target frequency lead to lock-in amplitude and phase images. On the right hand side a space resolved amplitude image calculated via pixelwise Fourier transformations from the data set in Figure 2 is shown (excitation frequency: 50 mHz).

A computer offering 64 GB RAM, 2 AMD Opteron 6172 processors with 24 CPU cores at 2.1 GHz and a Nvidia GeForce GTX 980 GPU was used to perform the data processing. Programming was done employing Matlab and its Parallel Computing Toolbox [9] on a Windows 7 operating system. We compare the performance of four different programming approaches: 1) a simple, non-optimized version; 2) a time optimized version considering Matlab’s peculiarities and strengths, using memory mapping and reducing unnecessary memory allocations; 3) a parallel computing version specifically adapted to Matlab’s ability to use up to 12 simultaneously computing CPU workers; 4) a version executing extensive calculations on the GPU.

a) Time Optimized Single-Thread Code
For the development of the time optimized single thread code, we took care to take advantage of Matlab’s peculiarities, namely considering the following points:
- Making use of vectorization instead of resorting to loops.
- Choosing appropriate data types. In our case the single-precision floating-point format proved as a reasonable compromise between accuracy and speed.
- Managing memory access and data storage according to Matlab’s internal procedures:
For the sake of comparison, the simple, non-optimized code was written without respecting those points.

b) CPU Parallel Computing
When working on multicore processors, Matlab is able to compute specifically written code on several workers simultaneously. Parfor loops (parallel for loops) are such a means to apply parallel computing to loops whose iterations are independent from each other. Furthermore spmd structures (single program multiple data) can perform the same blocks of code simultaneously on different data. Contrary to parfor loops, spmd structures allow an exchange of information between the individual workers and hence enable a parallelization of partly dependent program parts.

For a parallelization of the shearographic image processing, we used an spdm structure for the calculation of phase difference images, filtering and demodulation (since filtering does not allow parfor loops and since parfor loops applied to the phase difference calculation and the demodulation did not lead to a significant speed-up) and applied a parfor loop to the Fast Fourier Transformation (which can be done independently pixel by pixel).

c) GPU Parallel Computing
Matlab enables parallel computing on graphics processing units via Nvidia CUDA. Loading regular arrays into GPU memory using the command gpuArray(), subsequent functions applied to the array are performed on the GPU.

3. RESULTS

Comparing the simple and the time optimized single thread code results in a speed-up factor \( su = \frac{T_{\text{simple}}}{T_{\text{optimized}}} = 9.6 \). Regarding the optimized code as a new reference, further speed-up factors considering CPU parallel computing with different numbers of workers can be found in Figure 4. As creating a pool of parallel workers and assigning tasks and data to the workers requires time, parallelization results in a shorter overall computation time only from a minimum of 4 workers on. However, regarding a single worker as base value, the speed-up is approximately proportional to the number of workers.

![Parallel Computing: Relative Speed-up](image)

**Fig. 4** Speed-up factors of parallel algorithms using 1 to 12 workers relative to the time optimized nonparallel code.

Figure 5 observes the temporal distribution of the individual program parts for the optimized code and the different numbers of workers. Since the parallel structures do not allow to analyze the duration of the code line by line, the tasks executed in the spmd structure (phase difference, filter, demodulation and saving of data) are merged under pre-processing. Comparing the distribution of the optimized code to the distributions of the parallel codes, a distinct reduction of the FFT’s relative time consumption becomes noticeable. Most time can thus be saved using the parfor loop. However, the relative amount of pre-processing time decreases with an increasing number of workers which reveals a more effective parallelization of the spmd structure for larger numbers of workers.
Fig. 5 Distribution of the computing time taken for the image processing and Fourier transforming of a stack of shearographic difference images. The column labeled “optimized” designates the time optimized but nonparallel code, the other columns represent parallel algorithms employing 1 to 12 workers.

Utilizing the GPU for parallelization, we achieve a speed-up factor of 12.2 compared to the optimized single thread code. While small additional amounts of time become necessary for the conversion of regular arrays to GPU-arrays and vice versa, all other relevant parts of the program profit from the GPU usage, especially the filtering process, the latter being a traditional application of graphics processors.

4. CONCLUSION

Summing up, Figure 6 displays the speed-up factors of the optimized codes compared to the simple code. The time optimized code (2) works 9.6 times faster than the simple code (1), already offering a significant advantage. For the numbers of workers tested, the speed-up of the parallel computing version using the CPU depends approximately linearly on the number of employed workers and ranges from 2.8 for 1 worker to 29 for 12 workers (3). An advantage over the “time optimized code” is thus only given for a minimum of 4 workers, because for lower numbers the time for distributing the individual tasks to the workers overweights the parallelization’s benefit. The highest speed-up is achieved by the code resorting to the graphics processing unit (3) which is 119 times faster than the simple code. Our results join other recent publications affirming the benefit of parallel computing for the processing of optical measurements whose data sets have become more and more extensive and thus more complicated to handle over the years [10].
Fig. 6 Comparison of speed-up factors for the different codes using serial (simple code (1), time optimized code (2)) and parallel computing (parallel, CPU (3); parallel, GPU (4)) schemes. The number of parallel workers is only relevant for the parallel CPU-computing whose efficiency depends linearly thereon.

ACKNOWLEDGMENT

The proposal project 19063 N by the research association “Fördergemeinschaft für das SKZ” was funded through the “Arbeitsgemeinschaft industrielle Forschungsvereinigungen e.V.” (AiF) within the scope of the program for the Promotion of Cooperative Industrial Research and Development (IGF) by the Federal Ministry for Economic Affairs and Energy (BMWi) due to a decree by the German Federal Parliament. We would like to thank them for their financial support. Furthermore we would like to thank the members of the project-related committee for the discussions. SKZ is part of and gratefully acknowledges support of the “Deutsche Industrieforschungsgemeinschaft Konrad Zuse”.

REFERENCES


