Journal of Real-Time Image Processing (2022) 19:517–527
https://doi.org/10.1007/s11554-022-01204-4

ORIGINAL RESEARCH PAPER

CPU and GPU real-time filtering methods for dense surface metrology using general matrix to matrix multiplications

R. Usamentiaga[1]

Received: 24 August 2021 / Accepted: 1 February 2022 / Published online: 18 February 2022

Abstract
Filtering is a required task in surface metrology for the identification of the components relevant for automated quality control. The calculation of real-time features about the surface is crucial to determining the mechanical and physical properties of the inspected product. The computation efficiency of the filtering operations is a major challenge in surface metrology, as current sensors provide massive volumes of data at very high acquisition rates. To overcome the challenges, this work presents different real-time filtering solutions comparing the performance on the CPU and on the GPU, using modern hardware. The proposed framework is focused on filtering techniques that can be expressed using a finite impulse response (FIR) kernel that includes the Gaussian kernel, the most common filtering technique recommended by ISO and ASME standards. This research work proposes variations of the double FIFO and double circular filters. The filters are transformed into a series of general matrix to matrix multiplications, which can be run extremely efficiently on different architectures. The proposed filtering approach provides superior performance compared with previous works. Additionally, tests are carried out to quantify the performance of the GPU in terms of data transfer and computation capabilities in order to diminish the penalty imposed by data transfer from main memory to the GPU in real-time operations. Based on the results, an efficient batch filtering technique is proposed that can be run on the GPU faster than the CPU even for small profile and kernel sizes, offloading this task from the host CPU for optimal system and application response.

Keywords Surface metrology · Real-time filtering · Laser profiling

1 Introduction

Surface metrology is a field of utmost importance in product manufacturing that deals with the characterization of surface topography [1]. The resulting topography provides information about regular and irregular patterns, roughness, waviness and dimensions of the inspected product, providing the deviations of the surface from its intended shape. These features are crucial to determining the mechanical and physical properties of the product, including friction or electrical conductivity, but also the overall finishing quality according to relevant standards. For example high friction can lead to faster wear and shorter lifetimes and uneven flatness results in heterogeneous rolled-product plastic deformations. Thus, the evaluation of quality control features ensures compliance with quality criteria, preventing failures and guaranteeing the serviceability.

Despite a great development of measurement techniques, surface metrology still presents a significant challenge [2]. Most measurement techniques are based on mechanical stylus profilers or noncontact optical instruments [3]. Stylus based methods are still the most common for surface topography. However, they are time-consuming, which is a significant limitation. Also, they require periodic maintenance, as they are affected by wear [4].

Noncontact optical instruments determine the surface topography of the inspected object through the intensity registered in a sensor [5]. There is a large number of methods for optical surface metrology, but in general they can be classified into two major categories, depending on whether they require triangulation or not. Methods that do not require triangulation are based on the physical nature of light, for example about how the light travels in time or space [6]. Interferometry is the most accurate measurement technology

✉ R. Usamentiaga
rusamentiaga@uniovi.es

[1] Department of Computer Science and Engineering, University of Oviedo, Campus de Viesques, 33204 Gijón, Asturias, Spain

in this group [7]. However, it has a limited field of view and range. Thus, this type of sensors is mostly used for micro-scale surface metrology. Time-of-Flight (ToF) based surface metrology greatly extends the field of view and range, but at the cost of significantly reduced accuracy [8].

Interferometry-based techniques are the best choice for extremely-high-accuracy in low range measurements, and ToF-based techniques are good for low-accuracy and large object sizes. The triangulation-based methods land in between in terms of accuracy and range [9]. Therefore, they are the preferred choice in many industrial applications [10].

There are many different configurations for triangulation-based methods, but in the most basic form they use one or more cameras and one emitter of structured light. The most common approach for automated surface inspection in industrial applications is a configuration based on a single camera and laser line projector. This method can achieve high measurement accuracy in one direction. Moreover, in manufacturing production lines the products to be inspected move forward along a roll path or on a conveyor belt. This relative movement between the product and the sensor allows the measurement of the whole surface measurement without swiping the laser line, producing a dense set of points (i.e., point cloud) that accurately represents the surface topography.

The assessment of surface topography requires removing unwanted elements from the measured surface geometry. A common approach is to define a reference line or plane about which deviations are measured. In general, the surface is assumed to be comprised of different wavelengths. The assessment requires the identification of the components relevant for the required analysis, which is achieved using a filtering operation. The filtering procedure performs the separation of the surface topography into components with different frequencies. The most common components are referred to as noise, roughness, waviness and form [11]. Filtering is a critical process for surface metrology to ensure the accuracy of the assessment of surface topography.

A large variety of filtering techniques are used in surface metrology. In all cases, the goal is to remove unwanted components from the analyzed surface. Depending on the particular application, this could require the elimination of the roughness from the waviness or viceversa. The most basic approach used to be implemented in hardware using a resistor–capacitor (RC) network. However, nowadays analog filtering has been replaced by digital filtering.

The computation efficiency of the digital filtering operations is a major challenge in surface metrology. Current sensors provide massive volumes of data at very high acquisition rates. For example, recent profilometers can measure profiles at an acquisition rate up to 10 kHz, with each profile containing thousands of measurement points. This type of sensors enable the extraction of dense topographic information from products in manufacturing lines that are moving very quickly, also without affecting the productivity. However, the high production speed and the massive volume of data generated requires extremely efficient filtering techniques, particularly if filtering is required to be applied in real-time with the manufacturing process. These extremely demanding real-time requirements are commonly found in automated quality control, where deviations from tolerance specifications of geometrical features need to be detected very quickly. Based on this approach, engineering workpieces are characterized online, which ensures they can performs its designed functions and the manufacturing process is operating according to the specifications. This minimizes downtime, and maximizes efficiency and overall product quality.

To overcome the challenges related to the filtering of high volumes of dense topographic data in real-time, this paper presents different real-time filtering alternatives comparing the performance on the CPU and on the GPU, using modern hardware. The proposed framework is focused on filtering techniques that can be expressed using a finite impulse response (FIR) kernel. This includes the Gaussian filter, which is the most commonly used filter in surface topography, recommended by ISO 16610 and ASME B46 standards for establishing a reference surface [12]. Other filters commonly used in signal processing, such as the windowed-sinc, can also be expressed using a FIR kernel. Moreover, these kernels can be easily transformed to meet specific requirements: low-pass, high-pass, band-pass or band-reject. The proposed approach can be used with any of these kernels, making it versatile, flexible, and suitable for a wide variety of applications. This research work analyzes the most commonly used platforms with different acceleration strategies: multicore CPUs and GPUs [13]. This work presents filtering techniques based on the most efficient methods previously described in the literature for real-time filters: the double FIFO filter and the double circular filter. The proposed filtering methods are transformed into a series of general matrix to matrix multiplications. This way, the proposed filtering approach takes advantage of the extremely efficient code developed in this field pushed by recent advances in deep learning models, where general matrix to matrix multiplications are the fundamental building block.

Hardware accelerators such as GPUs tend to be superior to multicore CPUs both in run-time performance and energy efficiency [14], but the advantages can be compensated by the penalty imposed by data transfer. This work also analyzes GPU performance and bottlenecks. Based on this analysis, an efficient batch filtering technique is also proposed that diminishes this penalty. This provides the opportunity for data filtering on the GPU in real-time even for small profiles, offloading the CPU from this task.

The remainder of this paper is organized as follows. Section 2 reviews filtering for surface texture analysis; Sect. 3 presents the proposed approach for efficient filtering in real-time and the batch filtering technique to overcome the penalty imposed by data transfers in real-time operations; Sect. 4 discusses the results obtained, including tests with real data; Sect. 5 reports conclusions.

2 Filtering in surface metrology

Surface metrology analyzes the form, waviness or roughness on the surface of a manufactured product. Filtering is used to distinguish relevant from irrelevant information. The most common method is based on the Gaussian filter. This particular type of filter is classified as a Finite Impulse Response (FIR) filter in digital signal processing [15]. This filter is usually implemented by convolving the input signal with the filter kernel, which is equivalent to a multiplication in the frequency domain. This later approach is only appropriate when the kernel is very large.

Different filtering techniques have been standardized in surface metrology [16]. The most common digital filtering techniques are the envelope filter [17], the Gaussian filter [18], the Gaussian regression filter [19], the spline filter [20] and the wavelet filter [21]. International standards describe these filters, their parameters and provide guidance on the use, including ISO 25178 [22], ASME B46.1 [23] and ISO 16610 [24].

In general, a signal x is created by sampling physical phenomena at regular intervals of time. In surface metrology, a signal is created by recording the topographic information at equal intervals in the plane of the surface, i.e., signals are sampled in the spatial domain. The sampling interval is important, as a value too small can lead to a large number of highly correlated data points while a value too large can result in highly uncorrelated data, losing crucial spatial information. Different methods to establish optimal sampling intervals have been proposed in the literature [25]. In surface metrology, the wavelength (λ) is expressed in length units (meters), and the spatial frequency ($\xi = 1/\lambda$) is expressed in cycles per meter. Different techniques can be used to create regular spatial sampling, such as encoders that detect the motion and trigger acquisition signals

The kernel used in the Gaussian filter, standardized in ISO 16610 part 21, follows a Gaussian function, as can be seen in (1), where i is the position from the origin of the kernel, and $\alpha = \sqrt{\ln 2/\pi} = 0.4697$ is a constant designed to provide 50% transmission at a cutoff wavelength λ_c.

$$h(i) = \frac{1}{\alpha \lambda_c} e^{-\pi(\frac{i}{\alpha \lambda_c})^2} \tag{1}$$

An alternative approach is the idealized filter windowed-sinc, which is calculated by taking the Inverse Fourier Transform of the ideal frequency response. The kernel is given by (2), where f_c is the normalized cutoff frequency. This function presents abrupt discontinuities at the ends, which create ripples in the passband and poor attenuation in the stopband. The solution is to multiply the kernel by a smoothing window, for example the Hamming or Blackman windows.

$$h(i) = \frac{\sin(2\pi f_c i)}{i\pi} \tag{2}$$

The kernel shape of the Gaussian filter and the Windowed-sinc smoothed with the Blackman are similar for small lengths, but tend to differ when the window length is larger.

These kernels represent low-pass filters, but they can be easily transformed into high-pass, band-pass or band-reject kernels, making them very flexible and versatile. For example, a high-pass filter kernel, h_{hp}, can be calculated from a low-pass filter kernel, h_{lp}, using (3), where N is the length of the kernel. Combining these filters, the separation of signals into relevant components is straightforward. The separation is based on the definition of different wavelengths cutoffs: λ_s to separate shorter wavelengths considered noise from roughness, λ_c to separate roughness from waviness, and λ_f to separate waviness and longer wavelengths describing the primary form.

$$h_{hp}(i) = \begin{cases} 1 - h_{lp}(i) & \text{if } i = N/2 \\ -h_{lp}(i) & \text{else} \end{cases} \tag{3}$$

Given a particular kernel h, a FIR filter of order N can be implemented using the convolution sum described in (4), where $x[n]$ is the input signal and $y[n]$ is the filtered signal

$$y[n] = \sum_{i=0}^{N} h[i] \cdot x[n-i] \tag{4}$$

When working offline, the execution speed of the filtering operation is not very relevant. However, in automated quality control, real-time filtering is required to measure product features during manufacturing. Different works have evaluated the performance of Gaussian filters on the GPU. A filtering approach using the GPU is presented in [26]. The proposed approach has a speedup factor up to 4.8. However, it is compared with an unoptimized Matlab implementation. In [27] another alternative approach is presented. Results indicate the CPU-based implementation has better performance on small images, but the situation is reversed when large images are used. The research only evaluates the performance with images, and it is not designed to work in real-time. In [28], a filtering procedure is proposed with multiple GPUs. The performance is compared with Matlab on the CPU, but it does not propose a real-time comparison. Results, again, indicate kernel launches in GPU present a

huge performance penalty for small images. In general, only with large data sets the penalty for data transfer is compensated. For small surface maps (512×512), even implementations based on two GPU based are slower than CPU implementations due the required data transmissions and kernel launches in GPU [29]. The CPU optimizations in all these works are not clear. A different approach is presented in [30], where the image is partitioned into blocks that are processed in parallel by modern GPU architectures. This is applicable to the parallel execution of 1D recursive filters. This approach can be applied to offline Gaussian filtering, reducing memory bandwidth over a sequence of recursive filters.

An optimized implementation for FIR filters on multicore CPUs is presented in [31]. This work proposes different strategies to filter height maps in real-time. Optimizations are based on efficient strategies to store previous samples in memory, CPU parallelism, SIMD instructions and cache-line friendly data structures. Results indicate the double FIFO (first in first out) is optimal data structure to store data in memory, achieving 40.58 μs per profile, 5000 times faster than the conventional spline filter. The proposed approach is not compared with GPU-based implementations.

3 Real-time filtering approach

The proposed filtering approach in this work is to design a procedure based on the most efficient methods previously described in the literature for real-time filters: the double FIFO filter and the double circular filter. However, rather than designing specific optimizations to run these methods on particular CPU or GPU architectures, in this work the methods are transformed into a series of general matrix to matrix multiplications (GEMM). In previous works, the implementation of the filtering techniques was optimized for particular CPU and GPU architectures based on specific parallel execution models. For example, in [31] the optimization is based on parallels execution on multicore CPUs taking advantage of SIMD instructions (single instruction, multiple data). In [28] specific CUDA kernels are implemented to run the filtering process on multiple GPUs. This work proposes a different approach much more flexible and versatile: every filter operation is transformed into a matrix multiplication. Then, highly efficient implementations of these operations are used to make the proposed filtering approach provide superior performance than previous works.

3.1 Efficient matrix operations

Matrix multiplication is a fundamental mathematical operations used in many fields. Thus, very efficient libraries have been implemented to run these operations. The defacto standard in high-performance computing is BLAS (Basic Linear Algebra Subprograms), which is a specification that describes a set of low-level routines and operations for performing common linear algebra operations, such as vector, matrix–vector, and matrix–matrix operations. Many vendors provide optimized BLAS implementations for their hardware. Two of the most commonly used implementations are the Intel MKL (Math Kernel Library), supporting Intel x86 32-bits and 64-bits CPUs, and cuBLAS for CUDA Nvidia GPUs. These two implementations provide highly efficient and finely tuned BLAS implementations for their processors. Many other implementations exists, such as rocBLAS for AMD GPUs using ROCm, or OpenBLAS for different CPU architectures including x86-64, RISCV, ARM64, MIPS or Sparc.

3.2 The double FIFO filter and the double circular filter

The double FIFO filter and the double circular filter are two possible alternatives to implement real-time FIR filters. Thus, they can be used to implement a real-time Gaussian filter for surface metrology. Considering an optical surface metrology method based on triangulation, where a single line projector is used and there is a relative movement between the sensor and the inspected object, a single height profile is acquired at a time. This profile contains a variable number of points that describe the height across the object. The point $x[i, j]$ represents the height of the profile i at the position j. The proposed real-time filter is applied to all sequences of point positions from $j = 0$ to $P - 1$, where P is the number of points per profile. The filtering operation is applied for each acquired height profile considering previous samples, producing a filtered profile with a delay that depends on the length of the kernel.

The double FIFO filter is a variation of the FIFO filter. The difference is that while the FIFO filter requires shifting the buffer every time a new profile is acquired, the double FIFO stores the sequence of profiles and only performs large block memory transfers periodically. This provides the opportunity for large data transfers rather than transferring single profiles. This way, memory access is more efficient and the execution time is reduced.

The procedure is illustrated in Fig. 1, considering profiles with 4 points, $P = 4$ from 0 to 3, and a filter kernel of size 5, $K = 5$ from 0 to 4. The number of rows in the storage matrix that contains the sequence of profiles is almost twice the size of the kernel: $K * 2 - 1$ (row 0 is a the top of the matrix); and the number of columns is equal to the number of points per profile. A profile is stored in one matrix row. The first profile is stored at row $K - 1$. Next profiles are stored by decreasing the insertion row, as can be seen in Fig. 1b and c up until the profile is stored at the first row of the matrix. The

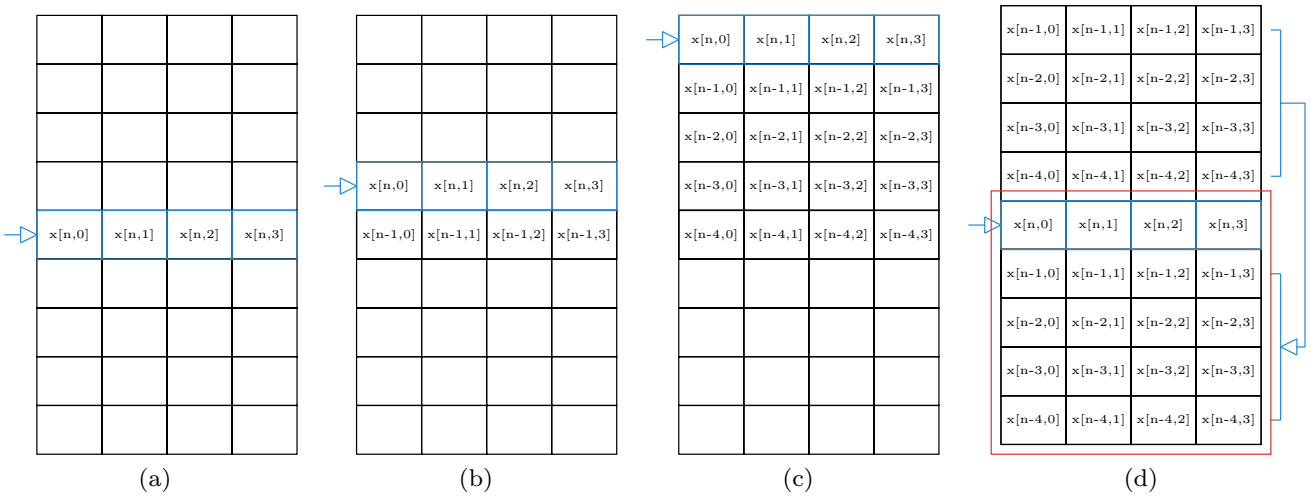

Fig. 1 Filtering using the double FIFO filter for $K = 5$. **a** First profile is stored at row $K - 1$. **b** Second profile is stored at position $K - 2$. **c** Fifth profile is stored at position 0. **d** The next profile is stored again at position $K - 1$ and the old data at the end of the buffer is overwritten with the most recent profiles

next profile wraps around the insert position to row $K - 1$. Also, the submatrix of profiles above the current position is copied to the submatrix below the current position, as can be seen in Fig. 1d. The profiles from the current position to that position plus $K - 1$ contain the block of profiles to be filtered (the most recent profiles). After the matrix is filled, a steady state is reached where the most recent profiles are always stored in a contiguous block of memory starting from the last insert position. Further details are given in [31].

The double circular filter is based on the circular filter. Similar to the double FIFO filter, this filter doubles the allocated size to store the previous profiles, maintaining two identical memory regions redundantly. This increased storage size is used to improve memory access.

Considering profiles with 4 points, $P = 4$ from 0 to 3, and a filter kernel of size 5, $K = 5$ from 0 to 4, the double circular buffer maintains two identical regions of memory. Thus, the storage matrix is $2 \times K$ rows and P columns. Two row indexes are used: the top index goes from $K - 1$ to 0, and the bottom index goes from $2 \times K - 1$ to K. Each acquired profile is stored twice, at the positions indicated by each index. When one index reaches the beginning of a region, it is wrapped around to the end of that region. In this case, no memory copy is necessary when one index is restarted. The most recent profiles are always stored in a contiguous block of memory starting from the top index.

The procedure is illustrated in Fig. 2. The first profile is stored at row $K - 1$ and $K \times 2 - 1$ redundantly. Next profiles decrement the position of both indexes, as can be seen in Fig. 2b and c. When the positions reach the beginning of each region, they are restarted to the initial positions and the process continues, as can be seen in Fig. 2d. Following this approach, at any moment the most recent profiles are always stored in a contiguous block of memory starting from the top index. Further details are given in [31].

Both the double FIFO and the double circular filters create a contiguous block of memory where the most recent profiles are stored. This block is a submatrix with a number of rows equal to the length of the kernel and the number of columns equal to the length of the profile ($K \times P$). The kernel can be multiplied by this submatrix to obtain the filtered profile, as can be seen in Fig. 3. The result of the matrix multiplication is the delayed filtered profile ($[1 \times K] \times [K \times P] = [1 \times P]$). A FIR filter has a linear phase response where the higher the order, the longer the time delay too. Thus, the resulting profile has a delay that depends on the length of the FIR kernel. When using a kernel of size K, the delay is $(K - 1)/2$. Following this approach, memory access is very efficient and the filtering procedure is transformed into a matrix multiplication applied in real-time with the profile acquisition.

3.3 The double FIFO batch filter

This work proposes an additional variation of the double FIFO filter: the double FIFO batch filter. In this filter, profiles are filtered in batches rather than one by one. This generates and additional delay. Now the delay not only depends on the length of the filter but also on the length of the batch.

A batch of profiles is created in main memory consisting in a tunable number of profiles. The batch is transferred to the GPU only when the batch is full, where it is stored as a contiguous block of memory in the storage matrix. Then, filtering is applied to all the profiles in the batch. Moreover, the size of the storage matrix is increased (only limited by

the GPU memory). This way, the number of data transfers is reduced while the size of the block transferred is increased.

The procedure is illustrated in Fig. 4 considering profiles $P = 4$ and a filter kernel of $K = 5$ and a batch of two profiles, $B = 2$. The batch consisting of two profiles is stored in the storage matrix, which contains previous profiles in ascending row order. Then the kernel is multiplied by the two highlighted matrices, resulting in two filtered profiles. As can be seen, the process can vary the batch size with no major modifications. The most recent profiles are still stored in a contiguous block of memory starting from the insertion index. However, in this case the data block transferred from main memory to the GPU is increased, improving the overall efficiency of the filtering procedure. The batch modification

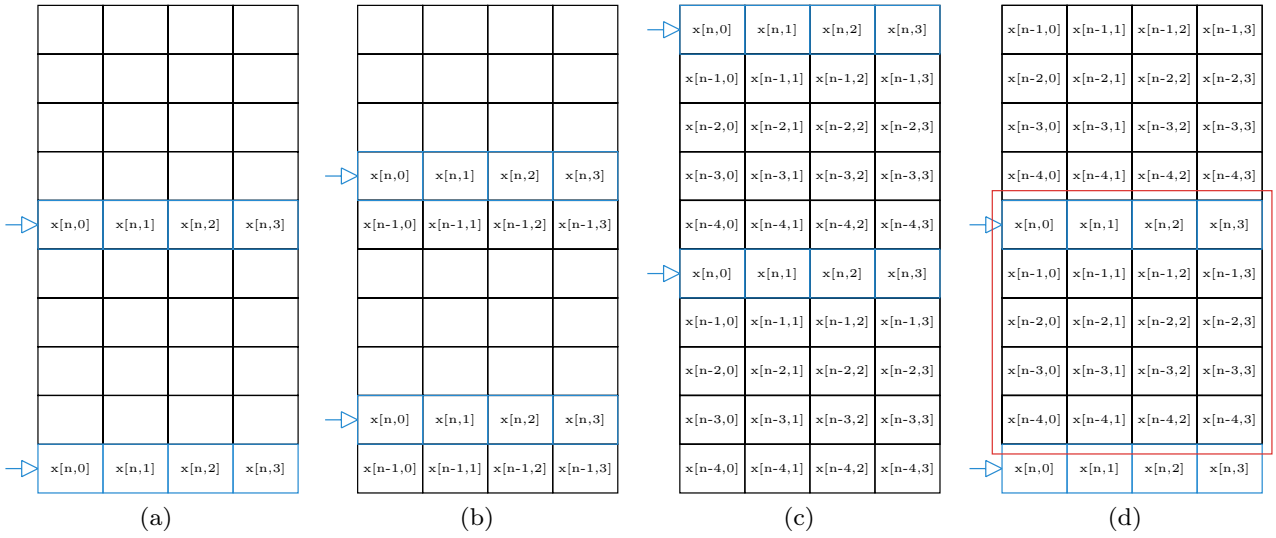

Fig. 2 Filtering using the double circular filter. **a** First profile is stored at row $K - 1$ and $2 \times K - 1$. **b** Second profile is stored at row $K - 2$ and the corresponding row in the bottom region. **c** Fifth profile is stored at the beginning of each region. **d** The next profile is stored again at position $K - 1$ and $2 \times K - 1$, and the old data is overwritten with the most recent profiles

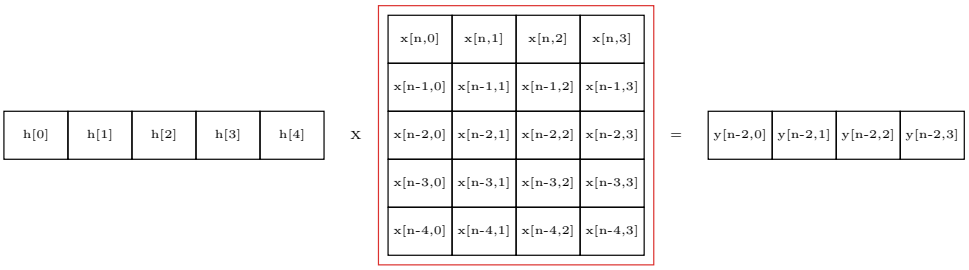

Fig. 3 Filtering using matrix multiplication: the kernel is multiplied by the submatrix containing the most recent profiles

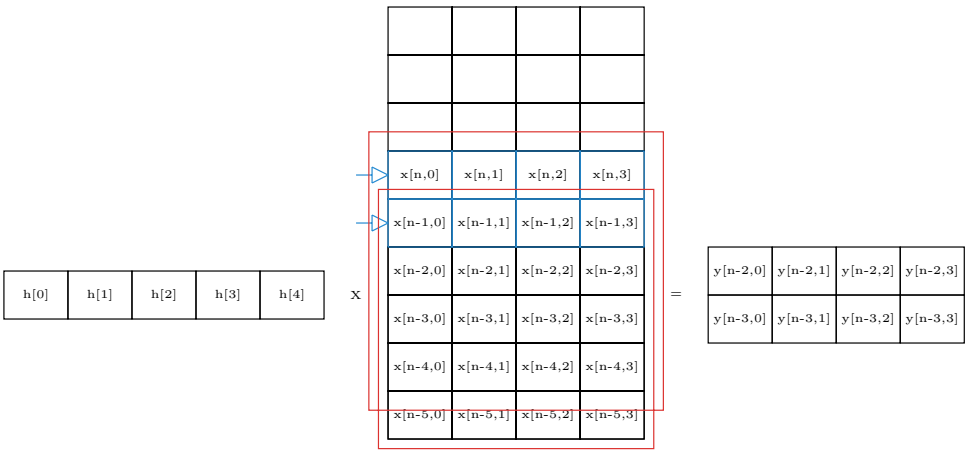

Fig. 4 Filtering using the batch filtering approach

is only applied to the double FIFO filter. Nevertheless, a similar approach could be used to create a batched version for the double circular filter.

4 Results and discussion

4.1 The double FIFO filter and the double circular filter

The real-time performance of the proposed filters are analyzed in this section. In all cases, the filters perform the same operation, providing the same results. The execution speed of each experiment is analyzed.

The most common scenarios found in surface metrology using triangulation-based methods are profiles with a length from 500 to 8000 points. Extended profile lengths are considered to evaluate the performance of the proposed approaches with future sensors. The considered kernel lengths go from 101 to 401, which could be valid lengths for usual low-pass filters of 400 mm when using a sampling interval of 10 mm.

Measuring execution time is a difficult but critical aspect to analyze the performance of the considered filters. The proposed testing procedure first perform a warmup while the storage matrix is filled. Then, the filter is run to find the minimum number of replicates that need to be run while still keeping measurement overhead low (to a small fraction of the overall run time). Finally, it runs as many replicates of the filter as required providing reliable estimates of the measurements. In addition, the testing procedure synchronizes the CPU and CUDA when benchmarking on the GPU.

All the reported experiments are performed on a computer with an Intel Core i7 9700K CPU running with 8 cores at 3.6 GHz and 64 GB of RAM. The computer also has a GeForce RTX 2080 Ti Turbo GPU with 11 GB of RAM. For the experimentation on the CPU the Intel MKL library is used [32], with optimized GPU GEMM implementations for Intel CPUs using all the available cores. When experiments are run on the GPU the cuBLAS library is used [33], which is the library that contains optimized GPU GEMM implementations for NVIDIA GPUs. All the implementations are compiled for x64 and the data type used to store the coordinates of data points is double-precision floating-point. All experiments are run under Linux 5.4.0.

Table 1 shows the time elapsed in the filtering of a single profile for the double FIFO and double circular filters on the CPU and GPU. The results show the execution time in μs for different values of K (kernel length) and P (profile length). There are two clear trends in these results: the double FIFO filter provides better results and the GPU provides better results only for large profiles or kernels. The double FIFO filter provides better results than the double circular filter in all cases, both when running on the CPU and the GPU. The double FIFO filter requires copying a block of memory once every K profiles. This approach results in better execution times than storing the profile twice each time, as the double circular filter does.

Table 1 Comparison of double circular and double FIFO filters running on the CPU and GPU. Al values are given in μs

K	P	CPU		GPU	
		Circular	FIFO	Circular	FIFO
101	500	15.4	11.2	30.7	21.3
101	1000	17.6	12.9	29.9	21.3
101	2000	20.7	16.0	31.1	21.8
101	4000	26.9	21.7	33.1	23.3
101	8000	41.8	36.3	37.5	28.6
101	16,000	178.8	192.8	66.3	56.7
101	32,000	660.5	656.4	100.1	89.9
201	500	18.9	13.3	29.8	20.4
201	1000	22.4	18.1	31.6	21.7
201	2000	29.1	22.7	31.3	21.6
201	4000	40.1	34.2	36.5	27.0
201	8000	153.9	158.2	52.7	43.0
201	16,000	610.9	586.4	90.4	80.3
201	32,000	1537.5	1517.1	145.8	136.2
301	500	20.8	17.4	30.8	20.8
301	1000	27.1	21.3	31.8	22.0
301	2000	36.2	30.0	32.7	22.6
301	4000	67.9	58.6	43.1	33.3
301	8000	351.4	347.6	62.7	53.2
301	16,000	1056.2	1082.6	111.8	102.1
301	32,000	2396.0	2393.8	188.8	178.7
401	500	23.5	18.8	31.0	21.0
401	1000	30.7	25.2	30.8	20.9
401	2000	42.9	37.3	35.7	26.1
401	4000	155.0	150.9	48.7	39.5
401	8000	591.7	585.3	74.5	65.3
401	16,000	1494.0	1480.4	136.9	127.3
401	32,000	3272.7	3259.8	234.9	224.7

Results indicate that the GPU only provides an advantage for large profiles or kernels. One of the main reason is the penalty imposed by data transfer from main memory to the GPU. As large data blocks are transferred and filtered this penalty is reduced. Only then, the GPU presents an advantage compared with the processing of the CPU. Moreover, GPUs implement efficient matrix operations by partitioning the matrix into tiles. In general, larger tiles use less bandwidth and result in more efficient data processing than smaller tiles. However, using larger tiles can generate fewer tiles to run in parallel, which can potentially lead to under-utilization of the GPU. The larger the matrix the GPU is operating on the less important this tradeoff is. When the

matrix is small tile efficiency or tile parallelism prevents the GPU from running at peak utilization. This issue can be seen in Fig. 5, which shows the double FIFO filter performance when considering the total number of points ($K \times P$). As can be seen, the GPU provides much better results than the CPU, but only if there is enough data to take advantage of the massive parallel execution capability of the GPU.

Comparing the results with previous works, for example in [31], the proposed procedure is around 4 times faster on similar hardware. These results confirm that the proposed approach to transform filtering into GEMM operations provides better results than directly designing ad hoc procedures based on SIMD and multicore parallelism.

The results presented in Table 1 indicate that for a kernel length of 101, a commonly used length, the GPU only presents an advantage when the profile has more than 8000 points. However, in most real scenarios profiles do not contain this large number of points. The most common profile length is around 2000 points. In this scenario, with $K = 101$ and $P = 2000$, the CPU is the preferred running platform considering the results of the experiments. The batch filtering approach is the proposed method in this work to improve GPU efficiency for this frequent scenario.

4.2 Analysis of the GPU performance and bottlenecks

To determine the best approach to performing batch filtering, several tests are carried out to quantify the performance of a GPU in terms of data transfer and computation capabilities. In addition, the penalty imposed by data transfer from main memory to the GPU is analyzed.

The first test analyzes data transfer speed from main memory to the GPU. Because the GPU is plugged into the PCI express bus, this largely depends on how fast the PCI express bus is and the number of lanes assigned. Other programs using the bus may also affect the results. The results are presented in Fig. 6. In this experiment, data is allocated in main memory and then it is sent to the GPU. Data is then transferred back to the host main memory. The GPU used in the experiment uses 16-lane slots (PCIe3 x16), which could give a theoretical 15.75 GB/s. However, the results provide worse results. Peak send speed of 11.36 GB/s is achieved when transferring memory blocks of 16 MiB. Peak gather speed of 10.11 GB/s is achieved also at 16 MiB. This test indicates that the most efficient approach is to transfer data in blocks of large sizes.

The next test evaluates the read and write speed in main memory compared with the same operations on the GPU. The results are presented in Fig. 7. The GPU can transfer memory much faster than the main memory. Moreover, the GPU can transfer memory much faster than it can get data from main memory. The peak GPU read and write speed is 691.09 GB/s at 2 MiB, i.e., it is around 70 times faster than data transfer from main memory to the GPU. The peak read and write speed in main memory is limited to 7.01 GB/s also at 2 MiB. This indicates that data transfer inside the GPU is

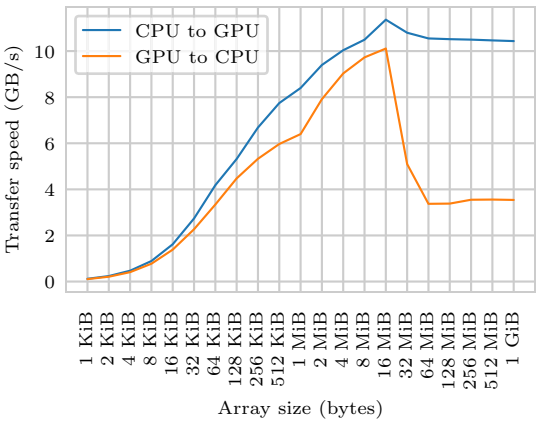

Fig. 6 Data transfer speed from main memory to the GPU and vice-versa

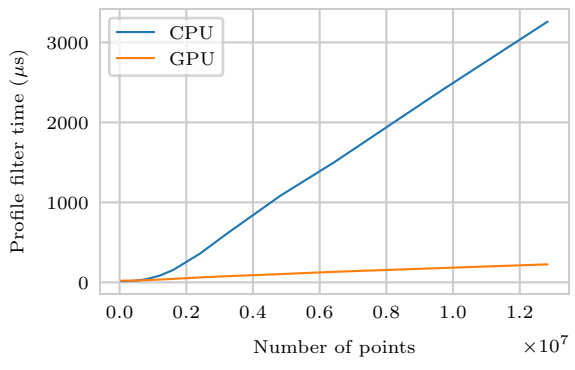

Fig. 5 CPU and GPU performance for profile filtering when considering the number of points

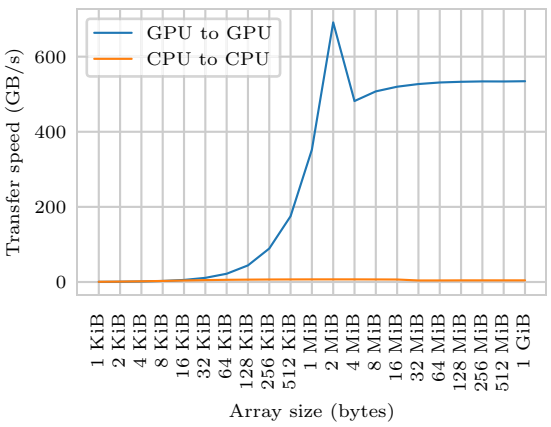

Fig. 7 Data transfer in the GPU and main memory

not a bottleneck, but the transfer from main memory. Therefore, to make filtering more efficient it is crucial to minimize the number of memory transfers from main memory to the GPU. Data transfer inside the GPU is not that relevant for efficient operations. This is the reason why the double FIFO filter is more efficient, as it is based on periodical data transfers of large blocks of memory inside the GPU.

The computational performance of the CPU and GPU is evaluated in Fig. 5. An additional test is performed to quantify the computational performance of a single matrix-matrix multiplication. For a matrix of *NxN* the total number of floating-point operations (GFLOPS) is calculated as $2N^3 - N^2$. The results can be seen in Fig. 8. The test achieved peak calculation rates of 921.2 GFLOPS for the CPU and 13207.1 GFLOPS for the GPU.

These results indicate that memory transfers from host memory to GPU memory is slow compared with data transfers inside the GPU memory. Thus, to improve the efficiently of the filtering process the number of transfers from the host memory to the GPU should be minimized, and they always need to be performed in large blocks. Also, when large data blocks are processed, GPUs are much faster than CPUs.

4.3 The double FIFO batch filter

Table 2 shows the results of the experiments for small profiles (1000 and 2000 points per profile). The results show the execution time per profile in the CPU and GPU in μs for different values of B (batch size) and $K = 101$. The benefit of using batches is clearly appreciable in the results, both in the CPU and the GPU. Also, when using a batch size of 75 profiles, the GPU is faster than the CPU even for this small profiles of 1000 points. For slightly larger profiles with 2000 points, a batch size of 8 profiles results in faster execution times in the GPU than in the CPU.

The comparison of the results in Table 2 with the results obtained previously in Table 1 shows that the execution in the CPU is worse when the batch size is small, but better

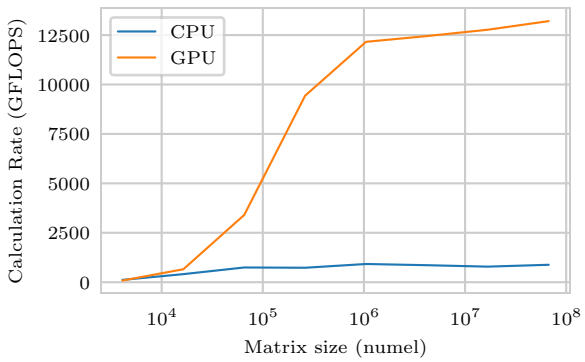

Fig. 8 Computational performance in single matrix–matrix multiplication in the CPU and GPU

Table 2 Results of batch filtering for $K = 101$ and different values of profile length (P) and batch size (B). Al values are given in μs

P	B	CPU	GPU
1000	1	17.8	25.4
1000	2	13.7	18.4
1000	4	13.9	14.4
1000	8	10.8	12.2
1000	25	10.9	10.8
1000	75	10.4	10.3
1000	150	10.2	10.2
1000	200	10.3	10.1
2000	1	20.8	26.2
2000	2	20.1	19.0
2000	4	14.0	15.0
2000	8	14.7	13.5
2000	25	13.1	11.9
2000	75	12.5	11.5
2000	150	12.6	11.3
2000	200	12.4	11.2

when the batch size is increased, as it is more efficient to transfer large blocks of memory to the storage matrix. Moreover, for a medium batch size the penalty imposed by data transfer from main memory to the GPU is negligible. Thus, this batch filtering approach can be applied in the GPU for any profile size. This approach offloads the filtering task from the host CPU resulting in optimal system and application response, as the CPU can run other tasks while the GPU is performing data filtering.

4.4 Real-time filtering for steel products

Surface metrology is applied in many different fields. Steel product manufacturing is among such fields where the surface finishing quality of the products is of crucial importance. Therefore, it is analyzed in real-time to assess the manufactured products.

Figure 9 shows the results of the proposed filtering approach applied to steel strips. The raw surfaces represent sections of steel strips 2 ms wide, 5 ms long and with height variations around 5 mm. The filtering results show the removal of the roughness and the waviness from the measured surfaces. This type of filters are very common in these products to analyze different features, such as flatness [34]. These filters are implemented using kernels calculated considering different cutoff wavelengths that distinguish relevant from non-relevant information. As can be seen, the filters efficiently remove the non-relevant information, generating the filtered surface in real-time with the manufacturing process. Acquiring the relevant information in real-time enables the extraction of features that indicate possible deviations of the surface from its intended shape, making quickly correcting actions possible and, thus, not

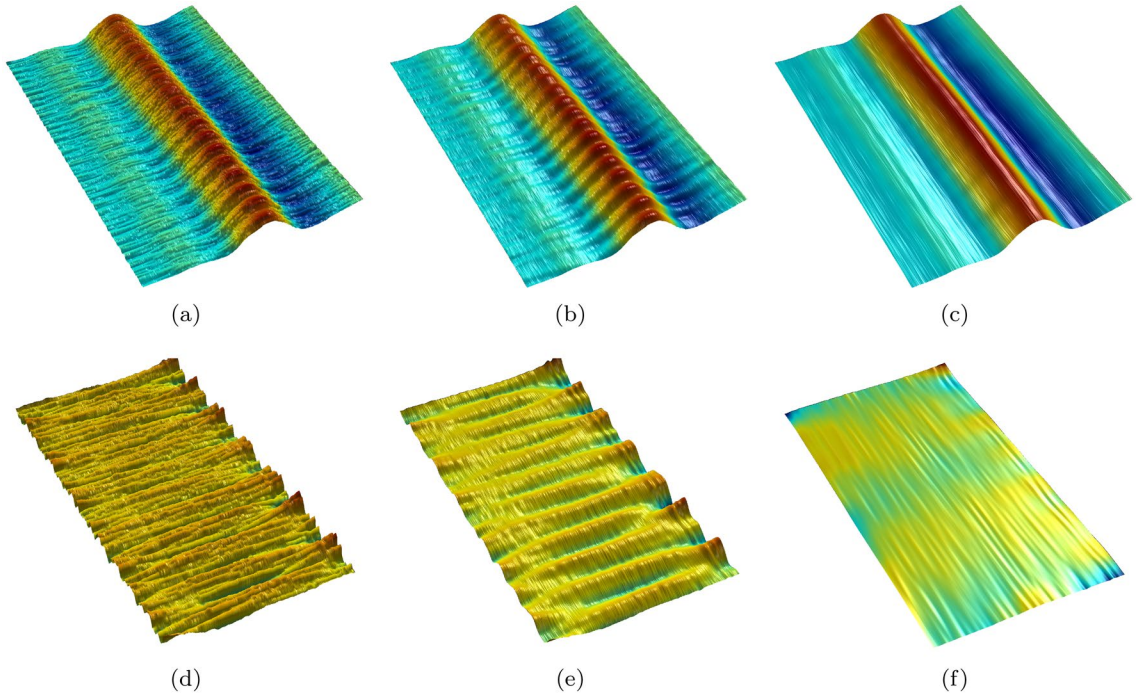

Fig. 9 Filtering results for steel strips. **a**, **d** Raw data. **b**, **e** Filtering results after removing roughness. **c**, **d** Filtering results after removing waviness

only improving the final quality but also the productivity of the manufacturing line.

In steel strip manufacturing, the most commonly used technique for surface metrology is laser triangulation based on line projectors. Profiles have from 1000 to 4000 points acquired at 1 to 4 kHz. The proposed filtering approach surplus these requirements. Thus, the integration of the filtering task in this industrial application is straightforward, even making it possible to run data filtering in low-end computers.

5 Conclusions

Real-time operations can be run on the GPU, providing superior performance and energy efficiency acceleration. However, in previous works the advantages are generally compensated by the penalty imposed by data transfer from main memory when processing small volumes of data. This work analyzes the performance of the CPU and the GPU for filtering operations required in surface metrology. The proposed approach is to transform filtering techniques into general matrix to matrix multiplications and, thus, take advantage of the extremely efficient code developed in this field pushed by recent advances in deep learning models.

Results indicate the proposed methods provide superior performance than previous works when running on the CPU. When filtering is applied on the GPU, the proposed methods provide major advantages for large profiles or kernels in terms of processing speed. However, small volumes of data are processed faster on the CPU, due to the penalty imposed by data transfer from main memory to the GPU. The solution proposed for this scenario, based on the analysis of the performance of the GPU, is batch filtering. Multiple profiles are transferred to the GPU in a single block of memory. Following this approach, filtering can be applied in the GPU for any profile size with better performance than the CPU. Therefore, the proposed batch filtering approach on the GPU not only provides an extremely efficient filtering method but also frees the CPU for other tasks. This represents a major advantage for a responsive real-time application. Furthermore, the proposed real-time approach is crucial for effective industrial applications, where engineering workpieces are characterized during the production process, which ensures they meet the required the specifications, minimizing downtime and maximizing efficiency.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

References

1. Gao, W., Haitjema, H., Fang, F., Leach, R., Cheung, C., Savio, E., Linares, J.-M.: On-machine and in-process surface metrology for precision manufacturing. CIRP Ann. **68**(2), 843–866 (2019)
2. Yi, B., Qiao, F., Huang, N., Wang, X., Wu, S., Biermann, D.: Adaptive sampling point planning for free-form surface inspection under multi-geometric constraints. Precis. Eng. **72**, 95–101 (2021)
3. Mathia, T., Pawlus, P., Wieczorowski, M.: Recent trends in surface metrology. Wear **271**(3–4), 494–508 (2011)
4. Conroy, M., Armstrong, J.: A comparison of surface metrology techniques. In: Journal of Physics: Conference Series, vol. 13, no. 1. IOP Publishing, p. 458 (2005)
5. Marrugo, A.G., Gao, F., Zhang, S.: State-of-the-art active optical techniques for three-dimensional surface metrology: a review. JOSA A **37**(9), B60–B77 (2020)
6. Townsend, A., Senin, N., Blunt, L., Leach, R., Taylor, J.: Surface texture metrology for metal additive manufacturing: a review. Precis. Eng. **46**, 34–47 (2016)
7. Wang, Y., Xie, F., Ma, S., Dong, L.: Review of surface profile measurement techniques based on optical interferometry. Opt. Lasers Eng. **93**, 164–170 (2017)
8. Corti, A., Giancola, S., Mainetti, G., Sala, R.: A metrological characterization of the kinect v2 time-of-flight camera. Robot. Auton. Syst. **75**, 584–594 (2016)
9. Beraldin, J.-A., Carrier, B., MacKinnon, D., Cournoyer, L.: Characterization of triangulation-based 3d imaging systems using certified artifacts. NCSLI Measure **7**(4), 50–60 (2012)
10. Usamentiaga, R., Molleda, J., García, D.F.: Fast and robust laser stripe extraction for 3d reconstruction in industrial environments. Mach. Vis. Appl. **23**(1), 179–196 (2012)
11. Muralikrishnan, B., Raja, J.: Computational Surface and Roundness Metrology. Springer, New York (2008)
12. Blunt, L., Jiang, X.: Advanced Techniques for Assessment Surface Topography: Development of a Basis for 3D Surface Texture Standards Surfstand. Elsevier, Amsterdam (2003)
13. Li, J., Peng, Y., Jiang, T.: Embedded real-time infrared and visible image fusion for uav surveillance. Journal of Real-Time Image Processing, pp. 1–15 (2021)
14. Georgis, G., Lentaris, G., Reisis, D.: Acceleration techniques and evaluation on multi-core cpu, gpu and fpga for image processing and super-resolution. J. Real-Time Image Proc. **16**(4), 1207–1234 (2019)
15. Smith, S.W.: The Scientist and Engineer's Guide to Digital Signal Processing. California Technical Pub, San Diego (1997)
16. He, B., Zheng, H., Ding, S., Yang, R., Shi, Z.: A review of digital filtering in evaluation of surface roughness. Metrology and Measurement Systems, vol. 28, no. 2 (2021)
17. Lou, S., Jiang, X., Scott, P.J.: Correlating motif analysis and morphological filters for surface texture analysis. Measurement **46**(2), 993–1001 (2013)
18. Young, I.T., Van Vliet, L.J.: Recursive implementation of the gaussian filter. Signal Process. **44**(2), 139–151 (1995)
19. Brinkmann, S.: Accessing roughness in three-dimensions using gaussian regression filter. Int. J. Mach. Tools Manuf. **41**, 2153–2161 (2001)
20. Krystek, M.: Discrete l-spline filtering in roundness measurements. Measurement **18**(2), 129–138 (1996)
21. Fu, S., Muralikrishnan, B., Raja, J.: Engineering surface analysis with different wavelet bases. J. Manuf. Sci. Eng. **125**(4), 844–852 (2003)
22. International Organization for Standardization: ISO 25178–2:2012 Geometrical Product Specifications. ISO, Standard (2009)
23. American Society of Mechanical Engineers: B46.1-2009 Surface Texture (Surface Roughness, Waviness, and Lay). ASME, Standard (2009)
24. International Organization for Standardization: ISO 16610: Geometrical product specifications (GPS) - Filtration. ISO, Standard (2002)
25. Dong, W., Mainsah, E., Stoutt, K.: Determination of appropriate sampling conditions for three-dimensional microtopography measurement. Int. J. Mach. Tools Manuf. **36**(12), 1347–1362 (1996)
26. Su, Y., Xu, Z., Jiang, X.: Gpgpu-based gaussian filtering for surface metrological data processing. In: 12th International Conference Information Visualisation. IEEE **2008**, 94–99 (2008)
27. Lee, C.W., Ko, J., Choe, T.-Y.: Two-way partitioning of a recursive gaussian filter in cuda. EURASIP J. Image Video Process. **2014**(1), 1–12 (2014)
28. Zhang, C., Xu, Y., He, J., Lu, J., Lu, L., Xu, Z.: Multi-gpus gaussian filtering for real-time big data processing. In: 2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA). IEEE, pp. 231–236 (2016)
29. Lustig, D., Martonosi, M.: Reducing gpu offload latency via fine-grained cpu-gpu synchronization. In: IEEE 19th International Symposium on High Performance Computer Architecture (HPCA). IEEE **2013**, 354–365 (2013)
30. Nehab, D., Maximo, A., Lima, R.S., Hoppe, H.: Gpu-efficient recursive filtering and summed-area tables. ACM Trans. Graph. (TOG) **30**(6), 1–12 (2011)
31. Usamentiaga, R.: Real-time filtering on parallel simd architectures for automated quality inspection. J. Real-Time Image Proc. **18**(1), 127–141 (2021)
32. Wang, E., Zhang, Q., Shen, B., Zhang, G., Lu, X., Wu, Q., Wang, Y.: Intel math kernel library. In: High-Performance Computing on the Intel® Xeon Phi^{TM}. Springer, pp. 167–188 (2014)
33. Markidis, S., Der Chien, S.W., Laure, E., Peng, I.B., Vetter, J.S.: Nvidia tensor core programmability, performance & precision. In: IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE **2018**, 522–531 (2018)
34. Usamentiaga, R., Molleda, J., Garcia, D.F., Bulnes, F.G., Entrialgo, J., Alvarez, C.M.S.: Flatness measurement using two laser stripes to remove the effects of vibrations. IEEE Trans. Ind. Appl. **51**(5), 4297–4304 (2015)