Computers and Electrical Engineering 40 (2014) 1158–1170

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Fast image processing for optical metrology utilizing heterogeneous computer architectures ☆

Marc Reichenbach *, Ralf Seidler, Benjamin Pfundt, Dietmar Fey

Department Computer Science, Chair of Computer Architecture, Friedrich-Alexander-University Erlangen-Nuremberg (FAU), Germany
Chair for Computer Architecture and Advanced Computing, Friedrich-Schiller-University Jena (FSU), Germany

ARTICLE INFO

Article history:
Available online 18 October 2013

ABSTRACT

Industrial image processing tasks, especially in the domain of optical metrology, are becoming more and more complex. While in recent years standard PC components were sufficient to fulfill the requirements, special architectures have to be used to build high-speed image processing systems today. For example, for adaptive optical systems in large scale telescopes, the latency between capturing an image and steering the mirrors is critical for the quality of the resulting images. Commonly, the applied image processing algorithms consist of several tasks with different granularities and complexities. Therefore, we combined the advantages of multicore CPUs, GPUs, and FPGAs to build a heterogeneous image processing pipeline for adaptive optical systems by presenting new architectures and algorithms. Each architecture is well-suited to solve a particular task efficiently, which is proven by a detailed evaluation. With the developed pipeline it is possible to achieve a high throughput and to reduce the latency of the whole steering system significantly.

1. Introduction

Image processing is a very complex and time-consuming task. Therefore, a well-considered choice of the applied architectures for the realization of algorithms is necessary. Often, only standard PCs with a serial processing scheme are used for the required tasks in industrial applications. This results in a low throughput and in high latencies.

However, image processing tasks are often easily parallelizable [1]. Taking a closer look at current CPUs, parallel computing is a rising market. While in 2006 only 2 cores (e.g. *Core2Duo*) were placed on a chip die, nowadays up to 8 cores work together as in the *Intel Sandy Bridge EP*. Additionally, the processors are internally extended via parallel computing units such as *Streaming SIMD Extensions (SSE)* or *Intel HT-Technology*. But are these new processor architectures sufficient for high-speed image processing algorithms? – It depends. Modern image processing algorithms, as used in optical metrology, contain numerous individual tasks with different granularities and complexities. While the raw data amount is quite high after capturing an image, the processing steps are very basic. On the contrary, postprocessing an image often requires complex floating point computation on small data packets. Hence, a standard multi-core CPU is not suitable for solving all these image processing tasks efficiently. Therefore, different architectures specialized for specific image processing tasks should be combined in order to achieve a high performance.

☆ Reviews processed and recommended for publication to Editor-in-Chief by Associate Editor Dr. Rene Cumplido.

* Corresponding author at: Department Computer Science, Chair of Computer Architecture, Friedrich-Alexander-University Erlangen-Nuremberg (FAU), Germany. Tel.: +49 91318527574.

E-mail addresses: marc.reichenbach@cs.fau.de (M. Reichenbach), ralf.seidler@uni-jena.de (R. Seidler), benjamin.pfundt@cs.fau.de (B. Pfundt), dietmar.fey@cs.fau.de (D. Fey).

0045-7906
http://dx.doi.org/10.1016/j.compeleceng.2013.09.008

A further problem is the high communication requirement in image processing systems. For high resolutions and high frame rates standard communication solutions such as Ethernet or USB are not sufficient. Other solutions, for example *camera link*, can achieve a higher bandwidth but require special and expensive frame grabber cards inside a PC. However, for real-time and high-speed image processing tasks with often more than 500 frames per second (FPS), these solutions are insufficient [2]. The only way to overcome the communication problem is to process or at least to preprocess the captured images close to the image sensor, directly inside the camera system. In many applications, a data reduction after the preprocessing in the image processing operation can be achieved. Thereafter, the data is small enough to allow a sufficiently fast transfer to postprocessing units by low-cost standard communication.

Therefore, for fast processing, heterogeneous architectures that are well-suited for the different tasks in the image processing operation are indispensable. A promising approach is the use of Field Programmable Gate Arrays (FPGAs) which allow the implementation of custom hardware architectures. Thereby, especially fine-grained tasks can be implemented as so-called processor arrays. Because of the low complexity of these tasks, a high degree of parallelism can be achieved. It is also possible to use soft-IP cores for FPGAs instead of custom designed hardware components. An example of a soft-IP core is a full-featured CPU offered in a hardware description language like VHDL, to be implemented in an FPGA or ASIC. This shortens the development time for the image processing system and allows a modification for individual requirements. Therefore, FPGAs are highly preferred for use in preprocessing captured images directly inside the camera – with added intelligence, the camera becomes a so-called *smart camera*. If the granularity of the tasks becomes more complex and, for example, floating point units are required, then GPUs are a better choice. Traditionally, these were designed for rendering geometrical objects, but the high degree of parallelism and the availability of fast floating point units make them valuable devices for accelerating complex tasks in image processing systems.

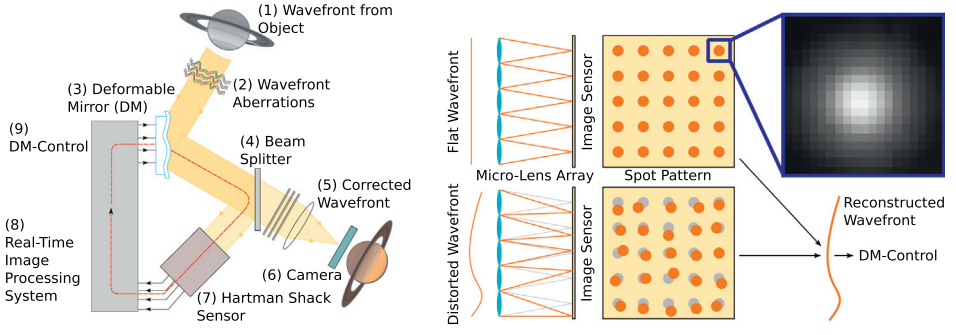
In this paper, the superiority of heterogeneous computer architectures is demonstrated using the example of an image processing pipeline, which consists of custom FPGA architectures, multicore CPUs and finally GPUs for fast processing. As an application, 3D wavefront measurements utilizing the Hartmann–Shack (HS) principle in optical metrology are used. This paper contains 5 sections. Section 2 explains the application of 3D wavefront measurements utilizing an HS sensor. Section 3 presents work related to this paper. In Section 4 our architectures and algorithms are presented, where we build the complete processing pipeline and evaluate its results. Finally, Section 5 entails the conclusion.

2. Hartmann–Shack sensor principle in adaptive optical systems

2.1. Hartmann–Shack principle

First of all, the optical metrology application will be presented in more detail. The basic idea is to measure 3D wavefronts utilizing an HS sensor for astronomic or ophthalmological applications. For our example, we used an astronomic application, which is used in a so-called *adaptive optical system* as shown in Fig. 1(a). The basic idea is to capture an image (6) from an examined object in space (1). Because of turbulences in the Earth's atmosphere, the image is aberrated (2). This problem is well-known in physics and called *wavefront aberration*. Hence, in the following we will call the light rays of such an object *wavefront* instead of image. To correct a wavefront (5), a deformable mirror (DM) (3) can be used, which requires an inverted wavefront profile. This is typically determined with the help of an HS sensor (7), which measures the derivation of the incoming wavefront. A beam splitter (4) is used to divide the wavefront between the sensor and camera (6). After the HS Sensor has captured the actual wavefront, the information is sent to a real-time processing system (8) where the input is processed. After the actual wavefront is reconstructed, its inversion can be calculated to control the DM (9).

Now we want to take a closer look inside an HS sensor as shown in Fig. 1(b). An HS sensor consists of an image sensor, such as a CMOS or CCD sensor. Instead of a normal lens, a so-called *micro-lens array* consisting of several lenses is placed in



(a) Example of an adaptive optical system with a Hartmann-Shack sensor.

(b) Hartmann-Shack sensor principle.

Fig. 1. Structure, functionality and application of a Hartmann–Shack sensor [3,4].

front of the sensor. The area of the image sensor, which corresponds directly to a micro-lens, is called a *sub-aperture*. If a planar wavefront reaches the micro-lens array, a spot is projected behind each lens on the image sensor, directly in the centroid of the sub-aperture. This spot is called the *reference point*. If an aberrated wavefront reaches the micro-lens array, the projected spots move away from the reference points. By measuring the aberration between a reference point and a captured point, the wavefront can be reconstructed. If the wavefront is inverted afterwards, a deformable mirror can be controlled to correct the telescope image.

Such an adaptive optical system is also called a *closed-loop* system because of the control loop between DM, HS sensor and real-time processing. In such a system hard requirements have to be met. A low latency of only a few milliseconds and a high throughput of several hundred FPS is required to work sufficiently. Moreover, the sensor resolution should be at least one mega-pixel with about 50×50 micro-lenses. The largest HS sensors currently available consist of 240×160 micro-lenses with an image sensor resolution of 16 mega pixel. But the problem of large image sensors is that they deliver only 3-4 FPS, which is not suitable for our intended fast image processing.

2.2. Processing tasks

Table 1 summarizes the processing tasks to be conducted. The basic idea is to measure the spot distance between the measured position and the reference point, thus determining the aberration. With that information, the measured wavefront can be reconstructed. Therefore, it is necessary to localize the captured spots in an image, which is done in **Task 1**, spot exploration. As shown in Fig. 1(b), each spot has a local, maximum gray value in the center. Since there is noise present in the image, it is necessary to introduce a threshold for the maximum. This is called spot detection (**Task 1a**) and works on integer data. Because of the necessity for maximal accuracy for the adaptive optical system, the spot positions are required in *sub-pixel* precision. This can be achieved because each spot consists of up to 9×9 gray valued pixels each. These values can be used to determine the spot centroids in sub-pixel precision (**Task 1b**) As a next step follows the so-called spot correlation (**Task 2**). For measuring the distance between the captured and reference points, it is mandatory to know which spot belongs to which sub-aperture. In most implementations for HS sensors the spots are not allowed to leave their corresponding sub-aperture, limiting the application to slight aberrations only. **Task 3** is responsible for the reconstruction of the wavefront based on the measured derivations in each sub-aperture. The reconstruction problem can be traced back to a least squares problem of the measured derivatives. This can be solved with iterative approaches such as the Jacobi method, the Gauss–Seidel (GS) method or the Successive Over Relaxation method (SOR), with increasing complexity and decreasing number of required iterations until convergence. **Task 4** delivers the reconstructed wavefront values, which are then visualized. Finally, the deformable mirror is controlled with this information.

As described in the section before, each processing task has its own granularity and complexity. Taking a closer look at Task 1, the whole image from the image sensor – normally a few megabytes – has to be processed. After this step, only the spot positions on the sensor are necessary for the successive tasks, requiring at most some hundred kilobytes. Moreover, the processing steps for Task 1 are very simple. For example, no floating point units are necessary and there are only local operations to perform. In contrast to Task 1, data amount of Task 3 is very small, but reconstruction of the wavefront is computationally intensive. Several hundreds of iterations on the whole input with considerable double-precision floating point operations have to be performed.

Therefore, we developed a heterogeneous image processing pipeline to solve all tasks as efficiently as possible and to achieve an increased throughput while decreasing the latency.

3. Related work

3.1. Smart camera architectures

One of the first smart cameras was developed by Wolf for gesture control [5]. Sousa developed a smart camera system for the application of static thresholding [6] which allowed for transferring only relevant data blocks utilizing data reduction. In [7] a high-speed camera with 500 FPS for edge detection, marker extraction and object tracking was presented. In [8] a low-cost smart camera was presented, which allows data compression inside FPGAs. This smart camera, with a Spartan-3E FPGA,

Table 1
Tasks for processing images utilizing an HS sensor.

Task	Name	Description
1	Spot exploration	Find spots in sub-pixel precision
a	Spot detection	Detect a spot with integer coordinates by finding the local maximum
b	Spot centroid calculation	Find the centroids of the detected spots in sub-pixel precision
2	Spot correlation	Match the spots found to the origin lenses where they are projected
3	Wavefront reconstruction	Reconstruct the wavefront with an iterative approach
4	DM control	Visualisation of the wavefront and controlling the deformable mirror

is now available through the company Elphel. Previously, we published a spot exploration architecture using this smart cameras for HS sensors [2]. There, the focus was on data reduction to save bandwidth between a camera and a PC.

For efficient FPGA-based processing inside a smart camera, special architectures are required. For example, spot exploration can be done efficiently with a streaming architecture. In [9] a generic VHDL template was presented which permits the fast processing of streaming applications. Parts of this architecture are also used in the image processing pipeline described here.

Furthermore, it is necessary to use special architectures for calculating object centroids. A memory-saving FPGA approach was described in [10] which uses so-called local moments. It does not require multipliers for finding object centroids in binary images. In [11] an ASIC architecture based on the local moment method was presented to find object centroids in grayscaled images. These methods were also applied to our pipeline.

3.2. Wavefront reconstruction in literature

Based on the method of Hudgin [12] there were some estimations on how to port the necessary algorithms to GPUs and FPGAs [13]. They implemented the spot detection with centroid determination (Task 1), as well as the reconstruction of the wave (Task 3) with a much simpler algorithm (the Jacobi method) than we did. For programming the GPU, the shader language OpenGL was used. Nowadays, CUDA or OpenCL are the first choice for programming GPGPUs in scientific applications. In their FPGA application a Virtex2 FPGA was used, where 32×32 sub-apertures could be reconstructed by applying 200 iterations within 0.317 ms. However, the much older GPUs were only able to compute the problem in 6.626 ms. The same group ported the FPGA implementation to a newer Virtex5 FPGA in 2010 [14]. There, they investigated the use of fixed-point representation instead of single precision floating point to improve speed while delivering the same precision in FPGAs.

However, the Hudgin method does not provide as promising results in terms of quality for the HS pattern as the Southwell method [15]. Although, the Jacobi method is easy to implement and easy to parallelize, it lags behind in its convergence rate. In our recent work, the SOR method as a reconstruction algorithm was implemented utilizing the much better hardware

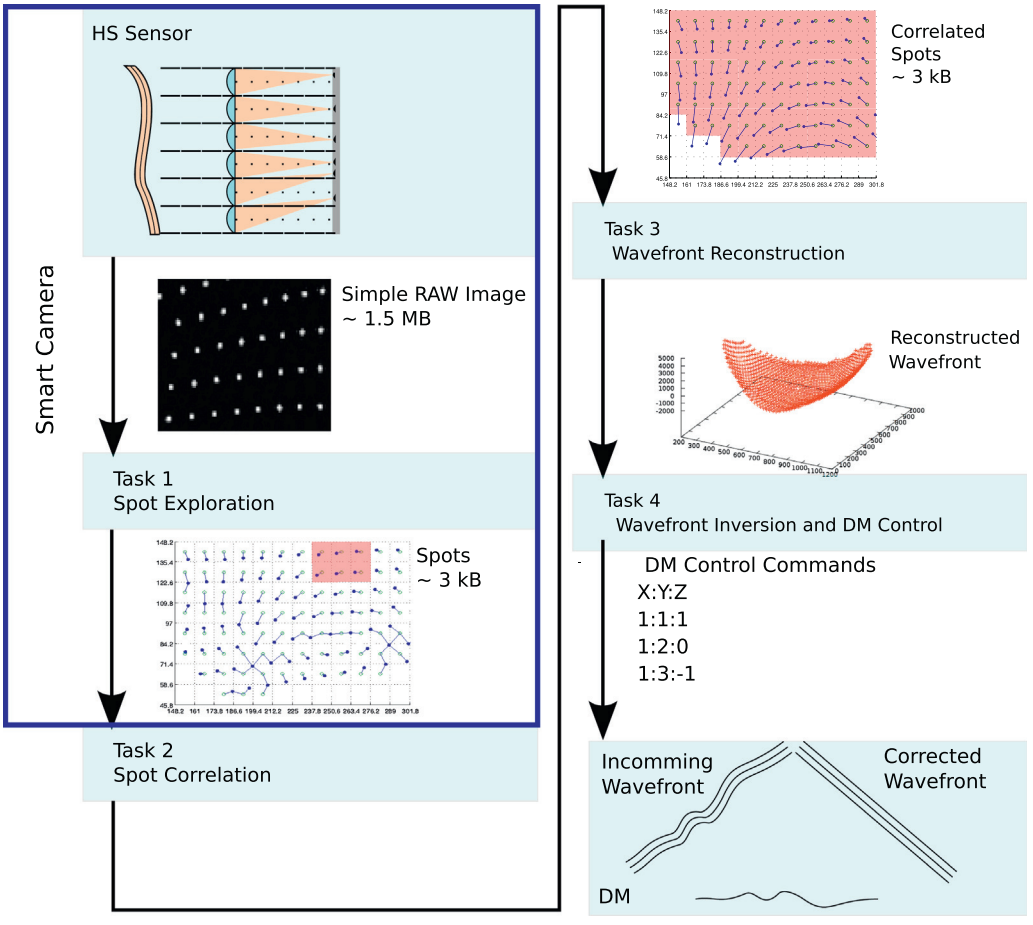


Fig. 2. Image processing pipeline with heterogeneous tasks. Finding optimal architectures required.

support for double precision of recent GPUs. This provides the ability to surpass the FPGA speeds with much better precision and an arbitrary grid size for the reconstruction task.

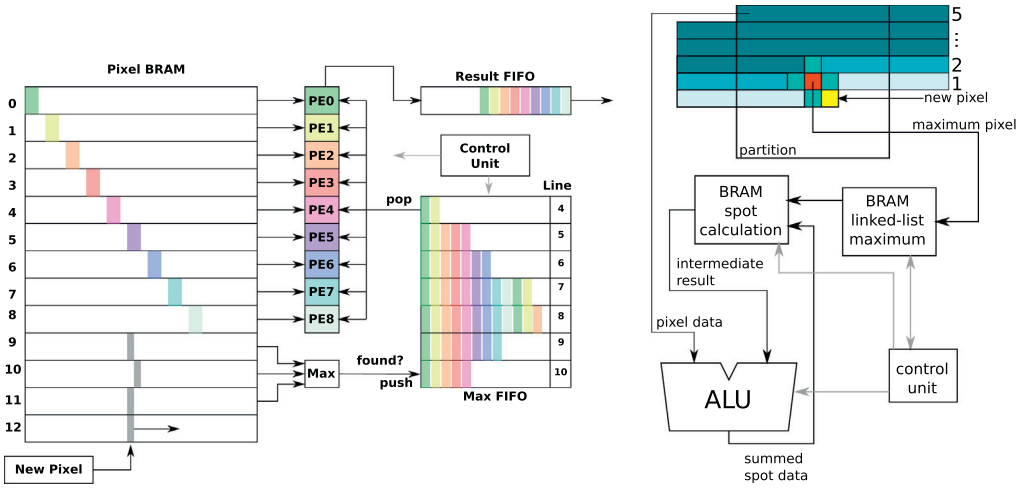
4. Architecture development and results

Fig. 2 shows an overview of the complete processing pipeline with the tasks described above. Previously, we presented a soft-processor (given as soft-IP) based solution for Task 1 [2]. In the following, we show various application-specific architectures that are much more efficient than the soft-IP based solutions. Furthermore, we present solutions for the other tasks of the pipeline shown in Fig. 2. As a result, the processing pipeline based on FPGA, CPU and GPU implementations allows us to apply the HS principle also in high-speed applications such as in adaptive optical systems.

4.1. Task 1 – Spot exploration

In our previous work [2] (called soft-IP approach) a soft-IP CPU array in an FPGA was realized to process Task 1 utilizing so-called ZPUs [16]. For this architecture up to 92% of the Spartan-3E FPGA resources were occupied. Because a frame rate of only 82 FPS at 82 MHz was achieved, we developed and evaluated some application-specific architectures, written in VHDL, for the spot exploration task in order to reduce resource consumption and to increase overall performance. In the following, the two developed architectures are presented and compared to the soft-IP solution presented in [2].

The first application-specific architecture we developed is based on an SIMD array (hence the SIMD approach) as shown in Fig. 3(a). The basic idea is to use the given architectural possibilities of a small scaled FPGA (such as the Spartan-3E) to deliver all found spots of a pixel row within one horizontal active phase of the sensor. For spot calculation, a partition of 9×9 pixels is required. Before the spot calculation in sub-pixel precision can be made, a maximum pixel above a specific threshold has to be found in the image. Then, it is possible to use the 9×9 pixels at the maximum pixel position to calculate the spot position in sub-pixel precision. To accomplish these tasks, a circular buffer of 13 image rows is used. A new pixel of the image is delivered every pixel clock from the image sensor and stored into a Block RAM (BRAM) inside the FPGA. A small processing element, *Max*, sweeps over the three last input-rows and looks for maximum pixels in a *von Neumann* neighborhood. At every processing step only three new pixels are required, since the left and current pixel have already been read in the previous step. If a maximum is found, its position is stored into a FIFO component from which the nine parallel SIMD processing elements (PE) can read. Here, a single FIFO for every processed image row is used. As can be seen in Fig. 3(a), row 10 is actually searched through for maximum positions, while the sub-pixel computing PEs work on row 4 as their middle point of the spots, using rows 0–8. Every PE works on a different image row of the nine rows. After each PE has finished its work on its row (summing up the pixel values for nine pixels and weighting them), they move forward by one image row in a circular way. For example PE zero obtains its computation with image row zero, PE one with row one and so on. After the computation has finished for each row, PE zero gets the data from row one, PE one from two and PE eight from image row zero. When a new image row is fetched from the sensor, the whole computation moves one row forward and the results are stored in another FIFO where the postprocessing tasks can be read. The mapping to FPGA resources is done by using BRAMs for the FIFOs and image rows. Because each BRAM can store only 16 kBit, we are limited to the small-scaled FPGAs in terms of image width and color depth, which results in a maximum resolution of 2048 pixel in 8 bit color depth per BRAM.



(a) Parallel SIMD processor element field. (b) Streaming architecture (MT).

Fig. 3. Different architectures for Task 1.

Taking a closer look at the SIMD based approach, the vast amount of employed logic resources, due to buffering, is evident. The optimal processing scheme for the spot exploration task would be to work directly on the incoming data stream. This would allow for a fast centroid calculation of the spot with minimum latency. Therefore, we implemented a second application-specific architecture, which is shown in Fig. 3(b). It realizes a streaming-based processing for Task 1 and is called multi-threaded (MT) approach. A 9×9 area around a found maximum is necessary for calculating the spot position in sub-pixel precision. Therefore, a buffer is required inside the FPGA, which stores all streamed pixel values as long as they are needed for processing. Only five complete image rows have to be buffered in the BRAMs of the FPGA which are used as large shift registers. All pixels from the sensor are stored and streamed through this buffer. The darker pixel with the *maximum pixel* output in Fig. 3(b) illustrates the pixel position in the buffer, where a maximum detection is made within a *von Neumann* neighborhood. After determining of the maximum, the coordinates of the 9×9 environment are stored in a linked list, which is also realized with BRAMs. Utilizing that information, the *control unit* can determine whether a pixel value streamed from the buffer is used for the actual spot calculation and controls the ALU. If so, the gray values are summed up and the spot centroids in sub-pixel precision can be computed. Therefore, a pixel is only stored in the internal buffer until it is clear what to do with its value. This leads to an optimal resource utilization. Since the pixels are sent in serial from an image sensor, the processing is also conducted in a serial fashion. This results in one ALU used by several register sets or spots, which are calculated simultaneously, as shown in Fig. 4. The approach is comparable to simultaneous multithreading (SMT) as applied in CPUs. The calculation of a spot can be seen as a thread, hence the name *MT*. The implementation of this architecture in a Xilinx Spartan-3E 1200 FPGA shows a maximum throughput of 132 processed FPS. It is evident that the most limiting resources in the FPGA are required for realizing the buffer and the linked list. All the results were created by the design flow with Xilinx ISE 12.

In Table 2 the results for the application-specific architectures are compared to the soft-IP-based solution for a one mega-pixel image with 255 sub-apertures. However, for the SIMD and MT architectures the number of sub-apertures is not a limiting factor, but the resolution of the used sensor and the FPGA type. Thus, it is easy to achieve 50×50 or 100×100 sub-apertures without restrictions. The architectures are based on the small-scaled Spartan-3E FPGA, which is available within the Smart Camera from Elphel. The results show that the application-specific architectures, first of all the MT architecture, are much faster and require only a fraction of the space inside the FPGA compared to the soft-IP based solution. These advantages arise at the expense of a decreased flexibility in the architecture. We also implemented the soft-IP design for a mid-sized Virtex-5 FPGA. With that it was possible to reach a frame rate of up to 500 FPS [2], due to a higher degree of parallelism and a higher clock frequency. However, the Virtex FPGAs are high-performance FPGAs that consume considerable energy, produce great heat and are expensive.

For energy efficient applications, Xilinx developed the Spartan-6 FPGAs as a replacement for the Spartan-3. The designs can benefit from the new device in three ways. First, through the new technology higher clock frequencies are possible, which allow for a higher throughput. Moreover, the chips contain more logic and BRAM resources. This enables the parallelization of our architectures to achieve a higher processing rate. Finally, some architecture limitations, for example the possible image width, as discussed above, can be extended. To use the new FPGA, a new Printed Circuit Board (PCB) shown in Fig. 5(a), was created with a Spartan-6 LX150 FPGA. Because of its small dimensions of only 54 mm $\times$ 45 mm it can be used within a smart camera. For compatibility reasons, the interfaces are the same as in the described camera from Elphel [8]. The PCB was designed using a six layer process and contains, in addition to the FPGA, a Linear LTM4616V as voltage regulator as well as a 24 MHz oscillator as a clock source for the FPGA.

The described architectures were enhanced for the new FPGA. For the MT architecture some improvements for internal precision, e.g. extending the width of the internal registers, were made. Table 3 summarizes the results of the implementation for a one mega pixel image with 100×100 sub-apertures. All architectures benefited from the new device because of the higher clock frequency and the large amount of available logic resources. Because the soft-IP approach is very flexible, the number of PEs was increased to utilize most of the FPGA resources for a high throughput. Compared to Table 2, the resulting FPS of the soft-IP approach decreased for the Spartan-6 FPGA. The reason is the larger problem size of 100×100 sub-apertures. Also, the application-specific architectures can be parallelized using multiple instantiations if the sensor can be read out in stripes as shown in Fig. 5(b). Because there are no data dependencies, except for the small range in the transition between the stripes, the throughput can be nearly doubled (marked by a in Table 3). Therefore, it is possible to achieve 300 or more FPS, depending on the image sensor.

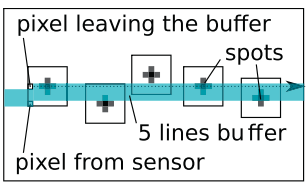
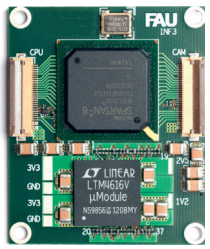


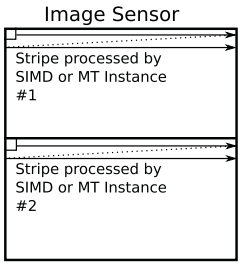
Fig. 4. Simultaneously computing different spots with only one ALU on a serial input stream.

Table 2
Achievable throughput and logic resource utilization for Task 1 for different architectures. Synthesis and PnR done with Xilinx ISE 12 for a Spartan-3E 1200 FPGA. Values in Brackets () are the maximum given by the FPGA type.

Architecture	Max freq.	BRAMs (28)	Register (17344)	LUTs (17344)	Util. (%)	FPS
Soft-IP	82	23	3876	15962	92	82
SIMD	118	15	2990	6688	53	118
MT	132	13	301	506	46	132



(a) Own PCB with a Spartan-6LX150 FPGA.



(b) Parallel processing utilizing SIMD or MT architecture.

Fig. 5. Performance improvements utilizing a new FPGA with more resources.

Table 3
Achievable throughput and logic resource utilization for Task 1 for different architectures. Synthesis and PnR done with Xilinx ISE 14 for a Spartan-6 LX150 FPGA. Values in brackets () are the maximum given by the FPGA type.

Architecture	Max freq.	BRAMs (268)	Register (184304)	LUTs (92152)	FPS
Soft-IP	102	136	26291	89686	36
SIMD	153	15	2874	5317	153[a]
MT	168	8	335	583	168[a]

[a] Values can be nearly doubled by a multiple instantiation.

4.2. Task 2 – Spot correlation

Spot correlation depends on complex floating point operations; therefore, we decided not to develop application specific-architectures for FPGAs. Instead, we investigated three general purpose architectures, which inherently provide FPU implementations. The first one is a standard multi-core processor from Intel, the *Xeon X5650* with six cores. The second is the *Leon 3* soft-IP processor developed and distributed by *Aeroflex Gaisler* [17]. It is written in VHDL and can be used and extended under the terms of the GPLv2. There exist several components such as an AMBA controller, an external memory controller or an FPU. The main advantage of the *Leon 3* processor is that a quad core system can be created. We also implemented the algorithm on a GPGPU, the *Nvidia Geforce GTX 480*. The details are presented in [18].

The algorithm for spot correlation has a basic idea. Assuming that a wavefront can change its curvature only slightly within a very small area, we can predict the approximated position of a spot by knowing the position of two correlated spots in the neighborhood. This situation is illustrated in Fig. 6, where the left spot and the spot in the middle are already correlated. By drawing a vector between the two spots and extending the vector itself, we can approximate the position of the next spot. At the approximate position, a search radius is defined in which we look for currently uncorrelated spots. The spot with the

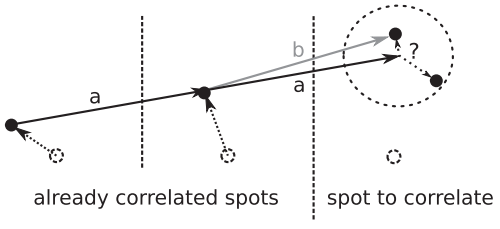


Fig. 6. Algorithm for spot correlation.

least distance to the predicted position is taken for correlation. For this approach it is important that at least two correlated spots are present. Because an image has two dimensions, a correlated area of 2×2 spots should be given. In the example of HS sensor, this can be achieved by declaring the spots in the center of the image as correctly correlated to their respective sub-aperture due to the least curvature of the lens in this region.

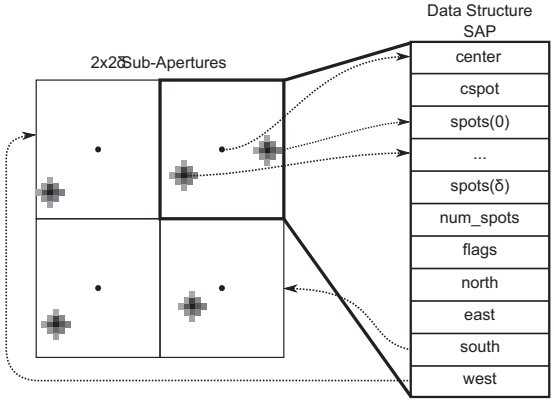
The implementation of the algorithm is based on a data structure for each sub-aperture, which is similar to [18]. The sub-aperture structure (*SAP*) is shown in Fig. 7(a). Each SAP contains the center of each sub-aperture (*center*) and a location for the spot to be correlated (*cspot*). The spot coordinates and their number inside each sub-aperture are derived from Task 1 (*spots* and *num_spots*). A *flag* indicates the current state of the SAP and the neighboring sub-apertures can be accessed by pointers (*north, east, south* and *west*). In a single core solution, one processor starts in the center of the image where spots have already been correlated. Based on these initialization points, the neighboring sub-apertures that can be correlated are stored in a queue. Successively, the elements are removed from the queue and the correlation algorithm is run. For each sub-aperture, the correlating spot, which can also be in a neighboring sub-aperture, is removed from *spots* and stored in *cspot*. New neighboring sub-apertures that have not yet been processed are appended to the queue.

Using parallelization for this task is a good choice to achieve maximum throughput. The simplest way is to use a common queue where multiple cores obtain sub-apertures to process and append the ones which can be used next. To preserve data consistency, mutual exclusions (*mutex*) have to be used. With more processors however, one list quickly becomes a bottleneck, because only one core can access the list at a time. Therefore, the algorithm was parallelized differently by performing geometric partitioning. A processing scheme with four partitions is shown in Fig. 7(b). Every core correlates only one predefined partition and starts with the sub-apertures at the center and ends at an image corner. To alleviate the problems with only one queue, each processor has a separate queue. If a sub-aperture in a different partition can be processed next, it is inserted in the queue of the corresponding core. Unfortunately, the queues themselves have to be protected by a *mutex*. But by taking a closer look at the algorithm, it can be seen that data dependencies between the partitions only exist at the edge areas. Sub-apertures further within the partition are only accessed by one core. Therefore, we introduced two queues per core: a synchronized queue for sub-apertures at the egdes and one for all other sub-apertures of the partition. To easily distinguish between the sub-apertures at the border, a special flag is set in the corresponding SAP.

To efficiently utilize the Leon 3 processors, no operating system was run. Because of this, a synchronization mechanism had to be implemented which is normally provided by an operating system. We based our mutex version on the Linux kernel implementation for the SPARC architecture. If the access is synchronized, values from other partitions can be read easily, because the Leon 3 processors have one unified shared memory.

In a second implementation the same code was optimized for an Intel CPU. The proposed algorithm fits perfectly to such high-performance CPUs, since all necessary data can be kept inside the large caches, due to the given FIFO principle.

The third implementation uses GPGPUs. Here, a FIFO principle is not feasible, because the data dependencies and lesser parallel execution limit the power of GPUs. We tried to achieve better results by using an iterative approach to determine the correlation for all sub-apertures. Here we need much more computation power and can harness the GPU much better. Nevertheless, there are not many parallel computations possible and the data access scheme does not fit in any way to that of SIMD GPGPUs. Thus, the results shown in Table 4 are as expected: the Intel CPU surpasses the other implementations. However, the FPGA soft-IP solution based on the quad core Leon 3 processor is able to accomplish this task directly inside the camera. But this is not suitable because of its low processing speed. To meet the real time requirements and achieve a high throughput, it is better to use the Intel CPU solution for Task 2. Hence, it is optimal to separate the processing between camera and PC at this task. This means that image capturing and Task 1 should be realized directly inside the camera. For the



(a) Structure for spot correlation in sub-apertures.

24	23	21	18	11	11	15	19	22	24
22	20	17	14	7	7	12	16	20	23
19	16	13	9	4	4	8	13	17	21
15	12	8	5	2	2	5	9	14	18
10	6	3	1	**A**	**B**	1	3	6	10
14	9	5	2	**C**	**D**	2	5	9	14
18	13	8	4	1	1	4	8	13	18
21	17	12	7	3	3	7	12	17	21
23	20	16	11	6	6	11	16	20	23
24	22	19	15	10	10	15	19	22	24

(b) Processing steps with four partitions and initialization points.

Fig. 7. Implementation for the Spot Correlation Algorithm.

Table 4
Execution times in milliseconds and resulting FPS in brackets () for Task 2, spot correlation, for different architectures depending on varying input sizes.

# Sub-Aper.	Leon 3 (4 cores)	Intel Xeon 5650	Geforce GTX 480
50	19.0 (52)	0.3 (3333)	2.5 (400)
100	106.0 (9)	1.4 (714)	5.5 (181)
150	174.3 (5)	2.9 (344)	10.9 (91)
200	268.8 (3)	5.0 (200)	19.3 (51)

other tasks, a PC should be used. The advantage of a separation at this point of the pipeline is that only a small data package per image, the information regarding the spot positions, have to be transferred to the PC which can be accomplished with low-cost communication hardware.

4.3. Task 3 – Wavefront reconstruction

Since we decided to execute spot correlation on standard multicore architectures, the following tasks are only implemented with standard processor architectures. Here a specialized hardware FPGA implementation is not feasible because the spot data has already been transferred to the PC.

The wavefront reconstruction is based on an iterative, least squares solution solver. We used the SOR implementation of [15] with a red-black pattern (called RB-SOR). This is important for a sufficient parallelization which allows for efficient implementation on a GPU or multicore CPU. We used the current GPU generation from NVIDIA (Nvidia Fermi) that has up to 16 independent vector processors (called *Multiprocessor*). Each of them has 32 SIMD processing elements with 48 kB scratch-pad memory (Shared Memory) and a 16 kB L1 cache. There exists no direct communication link between the multiprocessors, therefore every data-exchange has to be done via the external global memory on the GPU. The GS and SOR methods were also implemented on GPUs in [19,20] utilizing red-black coloring of the grid. Different grid sizes were examined, arriving at the conclusion that small grids with less iterations are not feasible for GPUs, e.g. in [20] a grid size of less than 200×100 (coarse grid) produces a speed-up of 1.1 when applying 500 iterations.

For a detailed evaluation of the solvers, the convergence rate and error functions of the wavefront reconstruction algorithm, described in [15], have to be examined. We chose a simple equation with well-defined derivatives. For that purpose we assume the function (1), with its derivatives given in Eq. (2) for direction x and y respectively.

$$f(x,y) = x^2 + y^2 \tag{1}$$

$$\frac{\partial f}{\partial x} = 2x, \quad \frac{\partial f}{\partial y} = 2y \tag{2}$$

We further assume an area definition $D = \{-1 \leqslant x, y \leqslant 1 \| x, y \in \mathscr{R}\}$, which is measured by $n \times n$ sub-apertures. In Fig. 8, the convergence rate for different $n = \{50, 100, 150, 200\}$ is shown for the GS, SOR and RB-SOR. The tolerance chosen was

$$\max(f_i(x,y) - f_{i-1}(x,y)) < 10^{-5} \quad \forall x, y \in D$$

where f_i is the value of the reconstructed wave at iteration i. Fig. 8 does not include the Jacobi iterations, since the convergence was not achieved within 10000 iterations. The results show that the GS algorithm is not feasible for our approach. Even for a small scaled 50×50 grid size, more than 1500 iterations are needed, where the other two algorithms stay below

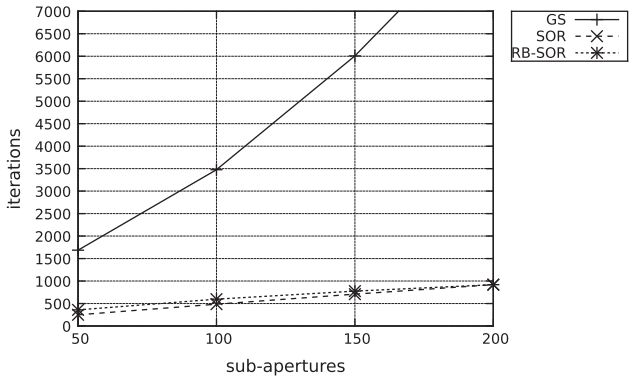


Fig. 8. Convergence rate for different solving methods and varying sub-apertures sizes.

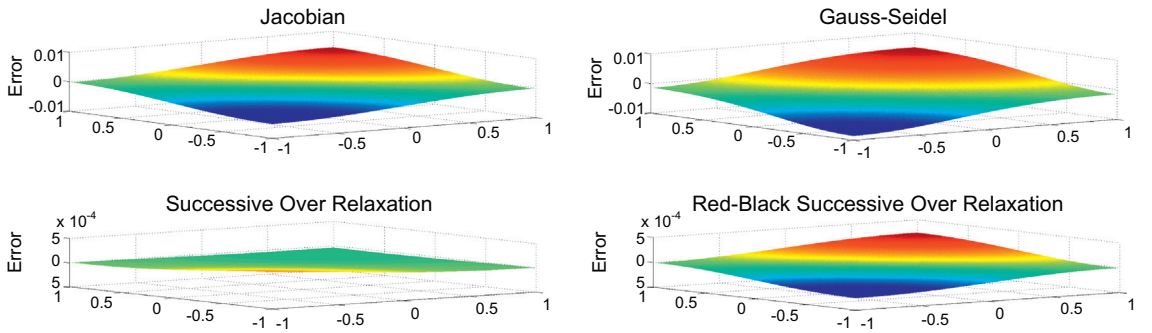


Fig. 9. Error function plots for different solving methods on a grid of 100 × 100 sub-apertures after achieving a convergence rate of 10^{-5}.

1000 iterations even for an area of 200 × 200 sub-apertures. A slower convergence rate for the RB-SOR can also be discerned for smaller applications. As the size of the application increases, the iterations become almost equal.

In Fig. 9, the resulting error is presented after the given number of iterations. It can be seen that by using the SOR method, the error is located in the foremost area, while using the other three methods the error is distributed among the field. Moreover, the error for SOR and RB-SOR are an order of magnitude lower than for Jacobi and GS.

Fig. 10 shows the maximum error function as well as the summed error for all four algorithms when applying 500 iterations for each. The Jacobi and GS are completely outperformed by the SOR methods and are therefore only listed here for completeness. An interesting fact is that the RB-SOR method has a better maximum error for larger applications and also a decreased summed error compared to the normal SOR method. Therefore, it is feasible not to check for convergence and always run the application up to 500 iterations.

To achieve a speed-up compared to a CPU implementation, our idea is to store all wavefront-values inside the GPU's shared memory. Thereby, a local synchronization strategy can be implemented in terms of a spin-lock-loop on every multi-processor, which has to wait for the others to finish their computation. Thus, no synchronization in the CPU space is needed.

The problem space of size $x \times y$ is separated into several stripes of size $x \times \frac{y}{\#mp}$, where each multiprocessor/thread-block works on one stripe and $\#mp$ denotes the number of multiprocessors. For a better performance, ghost-zones g are introduced on the top and bottom edge of the stripe, with a size of $x \times g$ each. The needed size of shared memory is given by

$$sm = \left[x \cdot \frac{y}{\#mp} + 2 \cdot x \cdot g \right] \cdot \texttt{sizeof(DATA\_TYPE)}$$

Each cell element can be float or double, denoted by `sizeof(DATA_TYPE)`. We chose $g = 3$, which allows the stripe to update four sub-iterations.

For simplicity, we assume that the wavefront is computed on a square shaped grid. The maximum number of cells for different GPUs are listed in Table 5. It can be seen that almost all GPUs have enough shared memory to store a grid of size 150 × 150 in double precision. Due to the limited amount of shared memory, the older Nvidia GPUs e.g. the Geforce GTX 280, based on the Tesla architecture, seems to be unsuitable for our needs. They can only update 8 (5 in double) rows per time-step and need to synchronize its 30 multiprocessors. Even the medium-scaled consumer card Geforce GTX 460 can easily outpower the GTX 280.

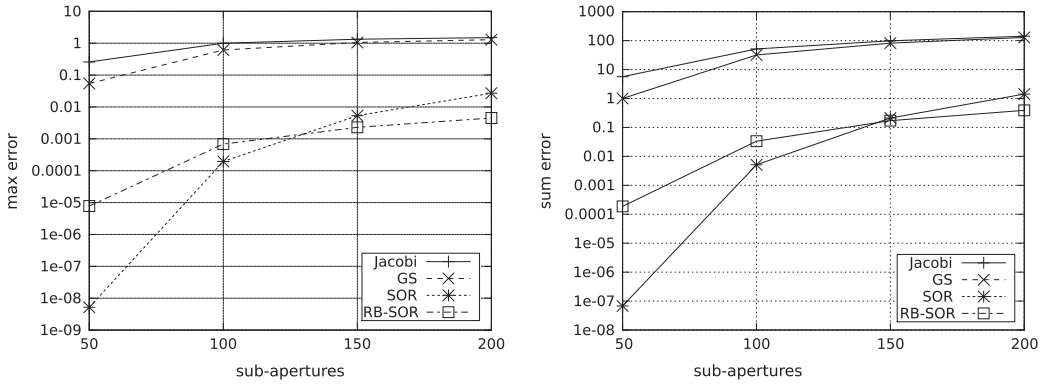


Fig. 10. Maximum and summed error functions for different solving methods and varying sub-apertures after 500 iterations in logarithmic presentation.

Table 5
Possible maximum grid sizes for the listed GPUs with float and double precision.

GPU	#mp	sm	DATA_TYPE	Cells/mp	Max (x)
GTX 280	30	16kB	Float	4000	256
GTX 280	30	16kB	Double	2000	155
GTX 460	7	48kB	Float	12000	268
GTX 460	7	48kB	Double	6000	183
GTX 480	15	48kB	Float	12000	379
GTX 480	15	48kB	Double	6000	255
GTX 580	16	48kB	Float	12000	390
GTX 580	16	48kB	Double	6000	261
Tesla C2050	14	48kB	Float	6000	367
Tesla C2050	14	48kB	Double	12000	247

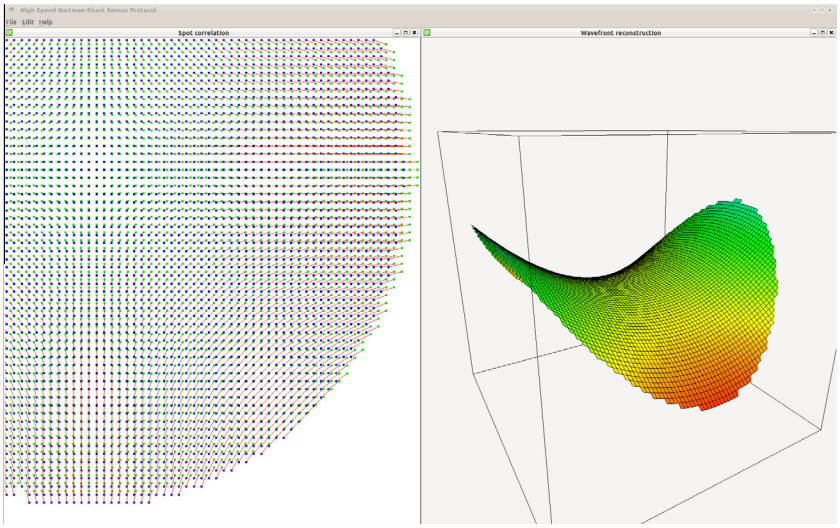


Fig. 11. Visualization of measurements in an open-source GUI application, written in Qt. 3D rendering is done by OpenGL-CUDA interoperability, 2D by native CUDA.

To synchronize the GPU's Multiprocessor cores, the GPU computation is normally invoked repeatedly by the CPU. This limits performance by repeatedly loading data from global memory. However, it is possible to invoke a busy wait on a microprocessor utilizing *atomic operations* on external memory. Using this method, a synchronization with the CPU can be eliminated. Another important fact is the reduction in the number of synchronizations by using additional ghost zones in overlapping areas between two multiprocessors. With such zones, the number of computations can be increased but the required synchronization points are reduced. As example, except for the first and last computation, every load and store operation is done utilizing shared memory.

For analysis, visualization and steering there are different closed-source software packages available on the market. For our purpose of developing an open-source HS-Sensor analysis toolbox, such closed-source software were not considered. Therefore, we developed an open-source software, where the measurement and steering of the spot exploration can be done seamlessly. It is written entirely in Qt and serves as a server application for the camera's client software. A screenshot is shown in Fig. 11. As shown, the program is divided into two main windows. The left one shows the measure spots (blue[1]) correlated to their respective sub-aperture (green). On the right side the 3D model of the reconstructed wavefront is printed, utilizing CUDA with OpenGL interoperability.

In Table 6 the required time for varying sub-aperture sizes are given for our GPU and CPU with a fixed number of 500 iterations. The times for the GPU implementation include the necessary communication overhead to transfer the spot data to the graphics memory and copy the resulting wavefront values back to the host memory. The results show that the GPU is the best choice for this task in our pipeline. We are able to achieve 500 FPS for a sub-aperture number of 50×50. However, the GPU provides another important possibility: it can work asynchronously with the CPU.

Therefore, we built our processing pipeline for a 100×100 lens HS sensor for the adaptive optical system. It has been implemented as follows with a resulting throughput of 294 FPS.

[1] For interpretation of color in Fig. 11, the reader is referred to the web version of this article.

Table 6
Wavefront reconstruction on GPU and CPU.

x	Geforce GTX 480		Intel Xeon 5650	
	Time in ms	FPS	Time in ms	FPS
50	2.0	500	3.1	322
100	3.4	294	7.4	133
150	5.3	188	14.6	68
200	8.0	125	24.8	40

1. Task 1 – spot exploration – is executed inside the camera with the presented MT architecture. Here we are able to reduce the data amount significantly, which simplifies the data transfer to a PC for postprocessing tasks. (up to 300 FPS, depending on the image sensor).
2. Task 2 – spot correlation – is executed on a CPU after spot data is transferred to the PC. (714 FPS).
3. Task 3 – wavefront recovery – is applied after spot correlation. This step is outsourced to a GPU, which is the most efficient architecture for the completion of this task. (294 FPS – In this pipeline the limiting factor).
4. Task 4 – steering the DM – at the end, the reconstructed wave needs to be transferred back to the CPU, where it is used for steering the deformable mirror.

While in our previous work [21] only a throughput of 132 FPS was able to be achieved, utilizing the new pipeline, now up to 294 FPS can be processed. Due to the increased performance compared to the standard solution, we would like to apply heterogeneous computer architectures also to other image processing applications. If similarities in the data flow and the granularity of the different processing tasks can be found in other applications, a speed-up should be attainable.

As discussed in Section 1, preprocessing tasks are mostly fine-grained with a high amount of data, whereas postprocessing tasks often work on a reduced data set with coarse-grained algorithms. Both tasks can be accelerated with adjusted parallel computer architectures. Currently we are investigating two applications with a similar task partition. The first application is Computer Tomography (CT) in material test. Near the sensor a noise reduction utilizing a bilateral filter has to be implemented on the raw data, which can be perfectly served by a PE-Array inside an FPGA. For the postprocessing step of the CT, the *Filtered Back Projection*, a coarse-grained architecture e.g. a GPU, is suitable. The second application is mobile scanning. There, a stereo image sensor is used to enable the recording of depth information for each image. It should be possible to implement the extraction of the depth information as a smart sensor. The reconstruction of a 3D environment afterwards is better suited in larger architectures.

5. Conclusion

In this paper, we presented an image processing pipeline for the Hartmann–Shack method which is used in adaptive optical systems. With this pipeline we were able to process several hundred frames per second in contrast to currently available Hartmann–Shack systems which only allow a throughput of a few frames per second. All presented architecture and algorithms were thoroughly analyzed and hand-optimized to achieve high performance. The throughput has been significantly increased by using a heterogeneous system architecture based on FPGAs, CPUs and GPGPUs. But the usage of such an architecture is not limited to this specific application: To increase processing throughput in industrial applications, many systems can benefit from heterogeneous architectures by using specific hardware components that are well-suited for the different image processing tasks.

Acknowledgment

The authors would like to thank the Deutsche Forschungsgemeinschaft (DFG) for funding this project (GRK 1773 – "Heterogeneous Image Systems").

References

[1] Bräunl T. Parallel image processing. Springer; 2001.
[2] Reichenbach M, Schmidt M, Nieszner T, Fey D. An image processing pipeline for fast spot detection in smart camera systems. In: Design & Elektronik (Org.): proceedings of the embedded world conference, Weka Fachmedien; 2012.
[3] Hippler S, Tokovinin A. Adaptive Optik Online am Max-Planck-Institut für Astronomie in Heidelberg. Max-Planck-Institut; September 2008. <http://www.mpia-hd.mpg.de/~hippler/AOonline/ao_online.html>).
[4] Adaptica, FAQ adaptive optics. <http://www.adaptica.com/site/en/pages/faq-adaptive-optics>.
[5] Wolf W, Ozer B, Lv T. Smart cameras as embedded systems. Computer 2002;35(9):48–53.
[6] de Sousa A. Smart cameras as embedded systems. In: Proena AJ, editor. 4th Internal conference on computer architecture; 2003.
[7] Mosqueron R, Dubois J, Paindavoine M. High-speed smart camera with high resolution. EURASIP J Embed Syst 2007.
[8] Filippov A. How to use free software in FPGA embedded designs. Xcell J 2003;46:74–7.
[9] Schmidt M, Reichenbach M, Fey D. A generic VHDL template for 2D stencil code applications on FPGAs. In: 15th IEEE international symposium on object/component/service-oriented real-time distributed computing workshops (ISORCW); 2012. p. 180–7.

[10] Komann M, Kröller A, Schmidt C, Fey D, Fekete SP. Emergent algorithms for centroid and orientation detection in high-performance embedded cameras. In: Proceedings of the 5th conference on computing frontiers. ACM; 2008. p. 221–30.
[11] Loos A, Reichenbach M, Fey D. ASIC architecture to determine object centroids from gray-scale images using marching pixels. In: Advances in wireless, mobile networks and applications, vol. 154; 2011. p. 234–49.
[12] Hudgin RH. Wave-front reconstruction for compensated imaging. J Opt Soc Am 1977;67(3):375–8.
[13] Marichal-Hernández JG, Rodríguez-Ramos LF, Rosa F, Rodríguez-Ramos JM. Atmospheric wavefront phase recovery by use of specialized hardware: graphical processing units and field-programmable gate arrays. Appl Opt 2005;44(35):7587–94.
[14] Díaz JJ, Dávila-González A, Rodríguez-Ramos LF, Rodríguez-Ramos JM, Martín Y, Piqueras J. Implementation of a waveform recovery algorithm on FPGAs using a zonal method (Hudgin). In: 1st AO4ELT conference – adaptive optics for extremely large telescopes.
[15] Southwell WH. Wave-front estimation from wave-front slope measurements. J Opt Soc Am 1980;70(8):998–1006.
[16] Zylin. ZPU architecture document; October 2011. <http://repo.or.cz/w/zpu.git?a=blob_plain;f=zpu/docs/zpu_arch.html>.
[17] Aeroflex Gaisler. GRLIB IP Core Users Manual; June 2011.
[18] Seidler R, Schäfer A, Fey D. Fast dot correlation in optical metrology on GPGPUs. In: Proceedings of the international conference on parallel and distributed processing technology and applications, vol. 1; 2011. p. 248–54.
[19] Courtecuisse H, Allard J. Parallel dense Gauss–Seidel algorithm on many-core processors. In: Proceedings of the 11th IEEE international conference on high performance computing and communications. IEEE; 2009. p. 139–47.
[20] Itu L, Suciu C, Moldoveanu F, Postelnicu A. GPU accelerated simulation of elliptic partial differential equations. In: 6th International conference on intelligent data acquisition and advanced computing systems (IDAACS), vol. 1; 2011. p. 238–42.
[21] Reichenbach M, Seidler R, Fey D. Heterogeneous computer architectures: an image processing pipeline for optical metrology. In: International conference on reconfigurable computing and FPGAs (ReConFig); 2012. p. 1–8.

Marc Reichenbach received his Diploma Degree in Computer Science at the Friedrich-Schiller University Jena in 2010. He works now as research assistant at Friedrich-Alexander University Erlangen-Nuremberg at the chair of Computer Architecture. His research interests are embedded systems, especially smart sensor architectures for different application fields.

Ralf Seidler received his Diploma Degree in Computer Science at the Friedrich-Schiller University Jena in 2010. He is employed as research assistant at the chair of Computer Architecture and Advanced Computing at FSU Jena. His research interests are high-performance systems and applied mathematics, with a focus on GPUs and other accelerator hardware.

Benjamin Pfundt studied Computer Science at Friedrich-Alexander University in Erlangen and received his Master Degree in 2013. He is now working in the PhD program ‘‘heterogenous image systems’’. His main research interests are smart image sensors and biosignal processing.

Dietmar Fey studied Computer Science at the Friedrich-Alexander University Erlangen-Nuremberg (FAU). From 1994 to 2001 he was researcher and lecturer at the Universities Erlangen, Jena and Siegen. In 2001 he became a professor for Computer Engineering. Since 2009 he holds the Chair of Computer Architecture at FAU. His research interests are parallel computer architectures, Cluster- and Grid computing and Nanocomputing.