Postprint: FPGA Implementation of the Lucky Imaging Algorithm

Authors: Zhao Panzi, Binhua Li, Mao Longhua, Tao Yong

Date: 2018-07-13T00:00:00+00:00

Abstract

Lucky imaging is a high-resolution image reconstruction technique employed to mitigate the effects of atmospheric turbulence on astronomical images. Conventional CPU-based lucky imaging algorithms are challenging to implement in real time. This work exploits the parallel processing capabilities of FPGA to design an FPGA-based lucky imaging algorithm and develop an experimental FPGA system. The system implements the complete lucky imaging algorithm pipeline —including frame selection, registration, and stacking—entirely on FPGA. The resulting high-resolution images are identical to those obtained through traditional CPU-based algorithms, yet the processing speed is 19 times faster than conventional CPU processing. The FPGA implementation of this algorithm offers a viable solution for real-time or near-real-time lucky imaging technology.

Full Text

Implementation of Lucky Imaging Algorithm Based on FPGA

Zhao Panzi, Li Binhua, Mao Longhua, Tao Yong
(Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming, Yunnan 650051, China)

Abstract

Lucky imaging is a high-resolution image reconstruction technique used to eliminate the influence of atmospheric turbulence on astronomical images. Traditional CPU-based lucky imaging algorithms are difficult to implement in real time. This paper leverages the parallel processing advantages of FPGA to design a lucky imaging algorithm based on FPGA and constructs an experimental FPGA system. This system completes the entire lucky imaging algorithm flow

—including image selection, registration, and stacking—using FPGA. The resulting high-resolution images are identical to those processed by traditional CPU-based algorithms, yet the processing speed is 19 times faster than conventional CPU processing. The FPGA implementation of this algorithm provides a feasible solution for real-time or quasi-real-time lucky imaging technology.

Keywords: Image processing; High-resolution image reconstruction; Lucky imaging; FPGA

Classification: TP391.4

The resolution of ground-based large-aperture optical telescopes for celestial imaging is limited by atmospheric turbulence [1]. To eliminate the impact of atmospheric turbulence on target imaging quality, common and effective methods include adaptive optics (AO) technology and post-processing image restoration techniques, among which lucky imaging is one prominent approach. Lucky imaging technology utilizes high-resolution structural information of targets contained in some short-exposure images to reconstruct images after observation, thereby eliminating atmospheric turbulence interference to obtain high-resolution images [1, 2]. The emergence of Electron Multiplying Charge-Coupled Device (EMCCD) imaging technology at the beginning of this century, with its advantages of high resolution, high readout speed, and low noise, is well-suited for low-light imaging applications with exposure times in the millisecond range and above [3]. This technological advancement has enabled significant progress in lucky imaging technology, achieving great success in astronomical observation, with over 200 papers published before 2013 [2]. Typical lucky imaging systems include LuckyCam from Cambridge University, AstraLux from the Max Planck Institute, and FastCam from the Institute of Astrophysics of the Canary Islands, which have produced abundant scientific results in the discovery of faint companions in binary or multiple star systems and in astrometric parameter calculations [4, 5].

Lucky imaging technology is a simple and feasible image restoration method. However, since image restoration is performed after observation, its disadvantage is also obvious: astronomical observers have limited real-time information about the captured images, making it difficult to promptly detect and correct possible deviations or errors in observation. One solution to this problem is to improve the algorithm and increase hardware processing capability [3], making lucky imaging technology real-time or quasi-real-time.

Traditional Central Processing Units (CPUs) process image data serially, making it difficult to achieve real-time or quasi-real-time performance for lucky reconstruction of large numbers of images. Compared with CPUs, Field Programmable Gate Array (FPGA) devices have inherent advantages in parallelism and flexibility, providing powerful parallel computing capabilities that can accelerate data or signal processing, particularly image processing [6], and are increasingly used in astronomical observation and data equipment [7]. Apply-

ing FPGA to lucky imaging is an effective method for real-time or quasi-real-time implementation of this technology. During the FastCam project research in 2008, lucky imaging technology was first implemented on FPGA hardware, achieving real-time high-resolution images [1, 8]; in 2015, Jackson also implemented the lucky imaging algorithm in FPGA and GPU [9]. However, they did not provide detailed descriptions of the FPGA implementation in their papers. No domestic reports on the combination of FPGA and lucky imaging algorithms have been published.

This paper briefly analyzes the traditional CPU and programming language-based lucky imaging basic algorithm, proposes a lucky imaging scheme suitable for FPGA processing based on FPGA technology characteristics, designs each functional module using Verilog Hardware Description Language (HDL), successfully implements the lucky imaging algorithm on an FPGA development board, and finally validates the feasibility of the hardware algorithm design using a large number of target short-exposure images captured by the 2.4m astronomical telescope at the Lijiang Observatory of Yunnan Astronomical Observatory, Chinese Academy of Sciences.

1. FPGA-Based Lucky Imaging Algorithm Processing Flow

The basic idea of the lucky imaging algorithm is to select "lucky images" or regions that meet certain evaluation criteria from a series of short-exposure images, and then perform registration and stacking processing to obtain high-resolution images unaffected or minimally affected by atmospheric turbulence [2]. The lucky imaging system constructed in this paper follows the basic design requirements of the lucky imaging algorithm during implementation. However, to adapt to the processing logic and resource constraints of the FPGA used in this system, the algorithm was redesigned using Verilog HDL without changing the basic processing flow (selection, registration, stacking).

1.1 Lucky Image Selection

Selecting "lucky images" from a series of short-exposure sequential images is the key factor for ultimately obtaining high-resolution images. To obtain "lucky images," an appropriate image quality evaluation function must be selected. For spatial point source target images, the Strehl Ratio (SR) is typically used as the evaluation standard. Since SR is defined as the ratio of the peak intensity of an image with aberrations to the peak intensity of an image without aberrations, calculating it requires an ideal aberration-free peak intensity, which is not easily obtained in practice. Therefore, alternative solutions are usually adopted. A simple and effective method is to use the instantaneous Strehl ratio as the image quality evaluation function [2, 10], which only estimates the SR value based on the gray value of the peak intensity pixel in the image.

For hardware like FPGA with limited computing resources, SR calculation can only use simple processing methods. Specifically, in this system implementation,

the gray value of the peak intensity pixel in the image with aberrations is used directly. Since the registration process has requirements for the position of the peak intensity pixel' s gray value in each frame, during the image selection process in this system, both the peak intensity pixel's gray value and its position are used together for image quality evaluation.

1.2 Image Registration and Stacking

Registration is the most important part of lucky imaging processing. Improper registration directly leads to blurred images after stacking, reduced resolution, and inability to display faint target images. Since this system uses stellar images, the most common registration methods are: (1) intercepting the required imaging region centered on the maximum gray value in the entire image, and (2) using the image centroid as the registration base point. Due to FPGA hardware logic and resource constraints, this system adopts the former registration method.

After image registration is completed, the registered images need to be stacked. Since the number of selected images in this system is relatively small, the direct stacking method is adopted. The resulting image then undergoes gray-scale transformation (i.e., sharpening) to obtain an easily observable high-resolution image.

2. FPGA Implementation of Lucky Imaging Algorithm

The hardware platform for implementing the lucky imaging algorithm described in this paper is a development board centered on the Xilinx Spartan-6 series XC6SLX16 chip. The development environment is ISE Design Suite 14.7, the HDL used for logic design is Verilog, and ChipScope Pro 14.7 and ModelSim 10.1d are used for hardware design verification and debugging.

2.1 Overall Algorithm Design

Since the main purpose of this project design is to implement the lucky imaging algorithm on FPGA hardware, astronomical short-exposure images are first stored in a Secure Digital (SD) card external to the development board for the algorithm module to read image data. Additionally, to meet the algorithm' s speed and on-chip storage requirements, and due to the limitations of the chip' s logic resources and storage space, this paper adopts a data flow approach—i.e., pixel-by-pixel processing—in terms of data processing methods. However, both within each module and between modules, data is processed in parallel. Meanwhile, to achieve optimal performance between processing speed and memory, on-chip dual-port Random Access Memory (RAM) modules are used as data buffers. However, due to constraints on RAM resources and logic resources of the FPGA chip used, only 1000 short-exposure images of $128\$\times128 pixel size can be processed for image selection, and during registration, only images up to 64\times64 pixel size$ pixel size.

For the entire algorithm' s hardware architecture, a modular design approach is adopted for convenience in modification and debugging during algorithm implementation. The main modules include data read/write and storage modules, short-exposure image selection, registration, and stacking modules, and the final reconstructed high-resolution image display module, all designed and implemented using Verilog HDL. The structural block diagram of the lucky imaging algorithm based on FPGA is shown in Figure 1 [Figure 1: see original paper]. The workflow of this lucky imaging FPGA hardware can be described as follows.

After system power-on, astronomical image data stored in the SD card is continuously transmitted to the SD card top-level module via Serial Peripheral Interface (SPI) at 6.25 Mb/s. The data processed by the SD card top-level module is continuously written into the DDR3 write data module via a First-In First-Out (FIFO) buffer (DDR3 speed 666 Mb/s). Then, through the processing of the DDR3 write data module, image data is written into the DDR3 memory external to the FPGA chip. While the write data module writes pixel data to DDR3, the image selection module also continuously solves for the maximum gray value of each frame. After all maximum gray values are sorted, image selection is completed, generating an image selection end signal. Subsequently, the image selection module sends the serial numbers of the original images to be intercepted and the parameters of their maximum gray value positions to the registration module, so that the registration module can send the first pixel addresses of the selected images to the DDR3 read data module. At the same time, the image selection end signal is also sent to the DDR3 read data module. When both the image selection end signal and the first pixel address of the selected image act on the DDR3 read data module, the module reads out the pixel values of the corresponding images from the DDR3 memory for stacking processing. Although the flow in Figure 1 shows that the registration, DDR3 readout, and stacking modules are serial, they are parallel when designed with HDL and implemented on FPGA, with their operations overlapping in time.

When stacking of all selected images is completed, the result undergoes piecewise linear gray-scale transformation to enhance the visibility of the target region. The final high-resolution image pixel data is then saved in on-chip RAM. After storage is complete, the gray-scale image is displayed on a VGA monitor through a Video Graphics Array (VGA) driver module.

2.2 Selection Module Design

In the implementation of the selection module, this system is mainly constructed using a maximum gray value solving module and a maximum gray value sorting module. These two sub-modules are primarily built using comparators, counters, and on-chip RAM storage.

The state transition diagram of the maximum gray value lookup module is shown in Figure 2 [Figure 2: see original paper]. In Figure 2, myvalid=1 indicates that data from the SD card can be retrieved; cnt represents the number of pixels

read from one frame, where cnt=0 indicates that all pixels of one frame have been read; eve_t=1 indicates that the maximum gray value of a single frame begins to be retrieved; pic_{num} represents the frame number; wea_{max}=1 indicates that data can begin to be written to RAM. As known from Section 3.1, the image size used in this design is 128$×$128 with 1000 frames, so the maximum cnt is 16383 and the maximum pic_{num} is 999.

As shown in Figure 2, the main function of this module is to compare each pixel value read from the SD card one by one to find the maximum gray value of each frame and save it in on-chip RAM for sorting. Notably, after obtaining the maximum gray value of each frame, it is first cached when triggered, and then when the corresponding condition arrives, it is judged whether it meets the registration requirements for the maximum gray value position. If it meets the requirements, the pixel value is saved in the corresponding RAM address space; if not, a zero value with the same bit width as the maximum gray value is stored in the corresponding RAM address space. Secondly, when saving the maximum gray value of each frame to RAM, the image serial number and position parameters of the maximum gray value must be simultaneously saved in the corresponding RAM address space for parallel-serial hybrid sorting.

2.3 Registration Module Design

In the implementation of the registration algorithm, the maximum gray value is used as the image center to intercept a 64$×$64 pixel region for stacking processing. The design mainly uses the image serial number where the maximum gray value is located and its position parameters sent by the selection module to calculate the first address of the image to be intercepted according to Equation (1), which is then sent to the DDR3 read data module to read the corresponding image pixel values from the DDR3 memory for the stacking module to process. This module does not occupy on-chip RAM resources, only using a small amount of Slices resources.

2.4 Stacking Module Design

Image stacking is mainly implemented through the use of on-chip RAM resources. In the stacking module design, the number of on-chip RAMs occupied is 12, approximately 37.5%. The block diagram of this module is shown in Figure 3 [Figure 3: see original paper].

3. Experimental Results and Analysis

To verify the correctness of the aforementioned lucky imaging algorithm implemented on FPGA, we plan to program the same lucky imaging algorithm in MATLAB and compare the processing results on a PC with those from FPGA. Regarding the processing time of the lucky imaging algorithm running on FPGA, it can only be compared with the running time on a PC. Therefore, this paper

designs two experiments: one is a comparison experiment with MATLAB processing results, and the other is the system' s own processing results and data.

Lucky imaging experiments require astronomical target short-exposure images. This paper uses observation images of the astronomical binary star HDS 70 from the Lijiang Observatory of Yunnan Astronomical Observatory on October 20, 2016, totaling 10,000 frames. The observation conditions and parameters of this image set are described in Mao Longhua et al.' s paper [10]. Regarding the selection of image frames and size, due to the limitations of the FPGA development board resources used, 1000 frames were randomly selected from the 10,000 frames multiple times and cropped to 128$×$128 pixel size for processing.

The experimental hardware platform is a Dell Precision T5500 graphics workstation with 16GB memory and NVIDIA GTX1050Ti graphics card, running Windows 7 operating system and MATLAB R2014a software. The randomly selected 1000 frames of 128$×128*pixelimageswereregrouped*, *andeachgroupwasprocessedwiththesamealgorithmi* pixel target region. According to the research results of Mao Longhua et al. [10], with a selection ratio of 1%, the high-resolution image and three-dimensional gray-scale distribution shown in Figure 4 [Figure 4: see original paper] were obtained. This result is consistent with Mao Longhua' s results.

3.2 Experimental Verification of Lucky Imaging Algorithm on FPGA

The hardware platform is a development board centered on the Xilinx Spartan-6 series XC6SLX16 chip. For verification of the lucky imaging algorithm implementation based on FPGA, a 1% selection ratio was adopted (i.e., selecting 10 frames from 1000 random frames), executing the same lucky imaging algorithm processing as MATLAB. Then, ChipScope was used to capture pixel data values around the maximum gray value in the final result image, obtaining the partial pixel data values of the high-resolution image processed by the FPGA-implemented lucky imaging algorithm shown in Figure 5(a), which were compared with the same partial pixel values of the high-resolution image processed by the MATLAB-implemented lucky imaging algorithm shown in Figure 5(b) [Figure 5: see original paper].

As shown in Figure 5, the final pixel data of the image processed by the FPGA-implemented lucky imaging algorithm is identical to the pixel data of the image processed with the same algorithm in MATLAB.

In terms of processing effect, the final high-resolution image pixel values are relatively small. When displayed directly through a monitor, the high-resolution images obtained from both MATLAB and FPGA algorithm processing are relatively blurry. Therefore, the same gray-scale transformation processing—sharpening or image enhancement—was applied to the high-resolution images obtained from algorithm processing on both FPGA and MATLAB, resulting in the final high-resolution images shown in Figures 6(a) and 6(b) respectively [Figure 6: see original paper].

Figure 6(a) was obtained by photographing with a mobile phone the high-resolution image processed by the FPGA-implemented lucky imaging algorithm displayed on an XGA monitor at 1024$\times$768 $resolution. Mobile phone photography was used because the FPGA de$ $resolution image obtained from the FPGA hardware-implemented lucky imaging algorithm must be displayed th$ $resolution image processed by MATLAB and displayed at 1280\times$768 WXGA resolution.

Due to the different monitor resolutions, the aspect ratios differ, causing slight display differences in the same image. Additionally, Figure 6(a) was obtained via mobile phone photography, and differences in shooting angle and resolution caused some variations in the image. However, the two images are still essentially identical, indicating that the results processed by the FPGA-implemented lucky imaging algorithm are consistent with those processed by the MATLAB-implemented lucky imaging algorithm, thereby demonstrating the correctness of the FPGA-based lucky imaging algorithm design.

In summary, whether comparing the pixel data values in the final high-resolution images or comparing the visually displayed final high-resolution images after gray-scale transformation, both demonstrate the correctness of the design and implementation of the FPGA-based lucky imaging algorithm.

3.3 Advantages of Implementing Lucky Imaging Algorithm on FPGA

The parallelism and flexibility of FPGA are its greatest advantages. Therefore, compared with traditional CPUs, FPGA definitely processes the same algorithm faster. However, for the lucky imaging algorithm studied in this paper, the specific speed improvement must be demonstrated through experiments. To this end, the processing speed of the same lucky algorithm was tested on both FPGA and CPU+MATLAB platforms. During the experiment, the processing time for 1000 frames of images was counted, with a selection ratio of 1%. Both FPGA and CPU+MATLAB platforms implemented the algorithm introduced in Section 2.1. The average running time obtained for different platforms is: 2.45s for FPGA and 46.93s for CPU+MATLAB. The results (high-resolution images) are shown in Figures 6(a) and 6(b) respectively. Obviously, the same algorithm produces identical high-resolution images on different platforms, but the algorithm processing speed on the FPGA platform is approximately 19 times faster than on the CPU+MATLAB platform. If more advanced FPGA devices are used, the speed will increase further.

However, constrained by the SD card speed, the overall running time does not have obvious advantages. Currently, we are designing high-speed data interfaces based on USB3 and Gigabit Ethernet to accelerate the entire lucky imaging processing speed. Additionally, using FPGA devices with stronger processing capabilities and more logic resources can introduce more parallel computing and reduce serial processing, fully leveraging the hardware acceleration advantages of FPGA.

Conclusion

This paper first briefly describes the basic ideas and processing flow of the lucky algorithm. Based on the hardware resources of the FPGA used, the key modules of the lucky imaging algorithm—image selection, registration, and stacking —were designed using Verilog HDL. The implementation scheme of these modules is introduced in detail, and this lucky imaging system was successfully implemented on an FPGA development board. Then, by comparing the FPGA processing results with those obtained using MATLAB on a PC, the correctness of the system design and implementation process was proven. Finally, through comparative analysis of the processing time required to implement the same algorithm on FPGA and CPU+MATLAB platforms, the 19-fold speed advantage of FPGA was demonstrated. Although this design scheme still has some areas for improvement, the specific implementation of the lucky imaging algorithm on the FPGA development board explores a feasible technical method for building a real-time high-speed lucky imaging system.

We thank Zhang Xiliang, Ji Kaifan, and Jin Zhenyu from Yunnan Astronomical Observatory, Chinese Academy of Sciences, for providing the original astronomical images for this research.

References

[1] Oscoz A., Rebolo R., Lopez R., et al. FastCam: a new lucky imaging instrument for medium-sized telescopes, Proc. Of SPIE, 2008, 7014: 701447.

[2] Mackay C.D., High-Efficiency Lucky Imaging[J], MNRAS, 2013, 432(1), 702-710.

[3] Hu B., Li B.. Electron Multiplication Model of EMCCD in Low Temperature[J]. Acta Electronica Sinica, 2013, 41(9):1826-1830.

[4] Law N.M., Hodgkin S.T., Mackay C.D.. The LuckyCam survey for very low mass binaries - II. 13 new M4.5-M6.0 binaries, MNRAS, 2008, 384(1): 150-160.

[5] Woller M., Bandner W., A Lucky Imaging search for stellar sources near 74 transit hosts, 2015, A&A, 579, A129.

[6] Treece B. CPU or FPGA for image processing: Which is best? [J], Vision Systems Design, 2017, 22(8): 23-24.

[7] Yin Q., Zhu L., et al. A New Method to Observe Radio Astronomy Signal: Centimeter-Decimeter Spectral Line—Based on High Performance Integrated Circuit. Astronomical Research & Technology, 2017, 14(1): 103-109.

[8] Piqueras J, Rodriguez-Ramos L, Martin Y, et al. FastCam: Real-Time Implementation of the Lucky Imaging Technique using FPGA[C]. Programmable Logic, 2008, Southern Conference on. IEEE, 2008:155-160.

[9] Jackson C R. Real time mitigation of atmospheric turbulence in long distance imaging using the lucky region fusion algorithm with FPGA and GPU hardware

acceleration[D]. University of Delaware, 2015.

[10] Mao L., Li B., Zhang X., Ji K., Jin Zh.. Experimental investigation of lucky imaging algorithm based on 2.4m astronomical telescope [J]. Opt Techn, 2018, 44(5) (In press) .

Note: Figure translations are in progress. See original paper for figures.