Optics Communications 316 (2014) 211–216

Contents lists available at ScienceDirect

Optics Communications

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

Wavefront processor for liquid crystal adaptive optics system based on Graphics Processing Unit

Dayu Li, Lifa Hu, Quanquan Mu, Zhaoliang Cao, Zenghui Peng, Yonggang Liu, Lishuang Yao, Chengliang Yang, Xinghai Lu, Li Xuan*

State Key Laboratory of Applied Optics, Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, Changchun, Jilin 130033, China

ARTICLE INFO

Article history:
Received 16 August 2013
Received in revised form
30 October 2013
Accepted 24 November 2013
Available online 6 December 2013

Keywords:
Adaptive optics
Liquid crystal
Graphics processing unit

ABSTRACT

To process real-time wavefront signal in our liquid crystal adaptive optics system (LCAOS), a real-time wavefront processor (RTWP) with graphics processing unit (GPU) accelerator is developed. A series of parallel acceleration algorithm is implemented to reconstruct wavefront, compute gray map and process lookup table (LUT). Especially, an algorithm by using symmetry of Zernike polynomial is designed to compute gray map and it provides a speedup of 3.16. Finally, the total computing time in our RTWP is 109 μs and temporal bandwidth in our LCAOS is 45 Hz; it is shown that our RTWP has the ability to real time process wavefront signal in our LCAOS.

1. Introduction

Adaptive optics (AO) technology is widely used in large aperture telescopes for optical astronomy. In recent years, liquid crystal spatial wavefront correctors (LCWFC) have become increasingly attractive as wavefront correctors in AO systems [1–7]. LCWFC has been investigated previously with promising results because of its many merits such as its high density of pixels, low cost, programmability, good repeatability, low power consumption, light weight, and so on. In the AO system, a real-time wavefront processor (RTWP) is used to detect the wavefront sensor (WFS) signals and compute the driving voltage signals that are to be sent to the corrector. This requires that all computing tasks to be completed within one frame. Previous systems have typically been using digital signal processors (DSP) and field programmable gate arrays (FPGA). However, an LCWFC has a large number of pixels so that wavefront reconstruction and gray map that is to be sent to the LCWFC would require a relatively high number of DSPs or FPGAs and extensive development time. Therefore, it will be unsuitable and costly for the liquid crystal adaptive optics system.

In recent years, graphics processing unit (GPU) has been used in extremely large telescope (ELT) AO systems with thousands of deformable mirror actuators [8]. The Palm-3000 AO system on the 5-m Hale telescope uses a 64×64 Shack–Hartmann WFS and a deformable mirror (DM) with about 3600 active elements, and is specified to have a maximum frame rate of 2 kHz [9]. For real-time wavefront processing, this system uses 16 Geforce 8800 GTX GPU, which obtains a computing latency of 205 μs. The Durham real-time controller (DARC) was initially developed to be used with the CANARY on-sky multi-object AO (MOAO) technology demonstrator [10]. It was tested that the performance of the DARC matched a wide range of proposed ELT AO systems by using GPU. Our liquid crystal AO system (LCAOS) on a 1.2 m telescope uses a 10×10 Shack–Hartmann WFS and a LCWFC with 65536 pixels, and is specified to have a maximum frame rate of 1400 Hz. Its computational requirement is about 13 million multiply–add operations in a frame. By contrast, the computational requirement of the Palm-3000 AO system is about 29 million multiply–add operations in a frame. In order to process wavefront in real time, our LCAOS also uses GPU acceleration.

The current wavefront processor based on GPU, such as the DARC, is a highly configurable AO control platform designed for the wavefront processing of current and as-yet-unknown AO systems. But it is suitable to control DM rather than to control the LCWFC. Moreover, our reconstruction and the control algorithm are different from the current AO system. Then the current wavefront processors based on GPU cannot be directly applied to our LCAOS.

Therefore, in this paper we report a modern real-time wavefront processor using GPU which would be used on our LCAOS. Our wavefront processing acceleration is programmed with the Compute Unified Device Architecture (CUDA) to reconstruct wavefront and compute gray map that are to be sent to the LCWFC in real

* Corresponding author. Tel.: +86 4 318 617 6319.
E-mail address: xuanli1957@sina.com (L. Xuan).

0030-4018/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.optcom.2013.11.052

time [11]. Section 2 presents an overview for the transfer function of our LCAOS. Section 3 presents an overview for the wavefront processing. In Section 4 we designed a parallel computing method for our computing algorithm, which is based on GPU to obtain the optimum performance. Finally, the results and conclusions are presented in Sections 5 and 6.

2. System transfer function overview

Our RTWP requirements are determined by the frame rate of the WFS, the number of Zernike modes and pixels on the LCWFC. The WFS's type is a Shack Hartmann and its CCD is the Andor DU860 CCD with a sub-aperture number of 100, an effective pixel number of 40×40 and a frame rate of about 1400 Hz. The first 209 Zernike polynomials on behalf of modal wavefront are used. The LCWFC is the BNS (Boulder Nonlinear Systems Corporation) PFP256 parallelly aligned liquid crystal spatial light modulator with 256×256 pixels, 90 μs data receiving time and approximately 714 μs voltage hold time of the digital analog converters (DACs) in order to match the frame rate of the WFS.

The phase transition of the LCWFC is approximated as

$$\varphi(t)=\varphi_0+(\varphi_1-\varphi_0)(1-e^{-t/\tau_{decay}}) \quad (1)$$

where φ_0 is the initial phase, φ_1 is the target phase and τ_{decay} is the decay time. Fig. 1 shows the phase transition (the wavelength $\lambda=785$ nm) of our LCWFC as a function of the time when the maximum voltage 5 v was released instantaneously at 0 ms. The blue cross points are the experimental phase transition values and the green solid lines are the theoretical fitted curve given by Eq. (1) when $\tau_{decay}=1$ ms. It is obvious that the theoretical fitted curve coincides with the experimental phase transition values. As shown in Fig. 1, the decay time τ_{decay} of the LCWFC is equal to 1 ms.

Fig. 2 gives a block-diagram representation of our open loop control algorithm. As shown, the uncompensated wavefront is denoted by φ_{tur}. The correction wavefront by the LCWFC is denoted by φ_{corr}. φ_{res} is the residual wavefront after correction, so that

$$\varphi_{res}=\varphi_{tur}-\varphi_{corr} \quad (2)$$

The main temporal characteristic of the WFS is the integration time T of the detector, so its transfer function is $WFS(s)=\frac{1-e^{-Ts}}{Ts}$, where $T=714$ μs. The main temporal characteristic of the RTWP is a pure time delay and a controller. In our open loop, the controller contains only the proportional gain, so the transfer function of the RTWP is $RTWP(s)=K_pe^{-(\tau_{ro}+\tau_{wp}+\tau_{ds})s}$, where $K_p=1$ is the proportional gain, $\tau_{ro}=714$ μs is the read out time, $\tau_{ds}=90$ μs is the time for data sending to the DACs and τ_{wp} is the wavefront processing time. In order to timely process each frame wavefront signals, the wavefront processing time τ_{wp} cannot exceed the integration time 714 μs of the WFS. The DACs hold the control voltages of the LCWFC until the next voltages are available from the RTWP. The transfer function of the DACs called a zero-order holder is $DAC(s)=\frac{1-e^{-Ts}}{Ts}$. Because the DACs are synchronized by the WFS, the time for voltage holding is the same as integration time of the WFS. The LCWFC temporal behavior is given by Eq. (1) and its transfer function is $\frac{1}{1+\tau_{decay}s}$. Finally, the residual error transfer function, denoted by $\varepsilon(s)$ is

$$\varepsilon(s)=\frac{\varphi_{res}(s)}{\varphi_{tur}(s)}=1-\frac{K_p(1-e^{-Ts})^2e^{-(\tau_{ro}+\tau_{wp}+\tau_{ds})s}}{T^2s^2(1+\tau_{decay}s)} \quad (3)$$

Once the wavefront processing time τ_{wp} is determined, we can simulate the temporal bandwidth f_{-3dB} for our LCAOS according to the residual error transfer function. Fig. 3 shows the simulated temporal bandwidth of our LCAOS as a function of the wavefront processing time. As shown, the wavefront processing time is as short as possible in order to improve the temporal bandwidth of our LCAOS.

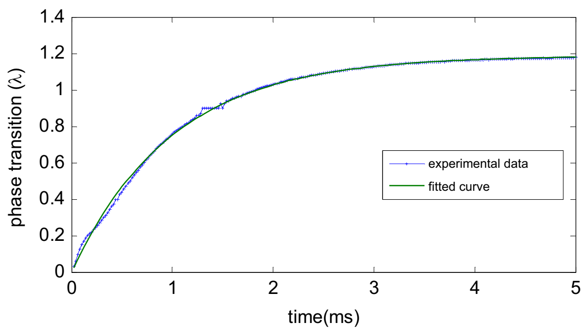

Fig. 1. Phase transition of our LCWFC as a function of the time when the maximum voltage 5 v was released instantaneously at 0 ms.

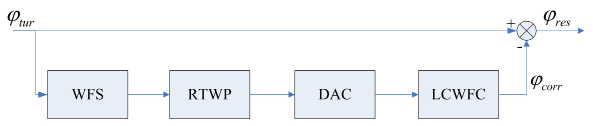

Fig. 2. Block diagram representation of our open loop control algorithm.

3. Wavefront processing overview

In the most recent AO systems, wavefront construction is a standard matrix–vector multiplication (MVM). The matrix in the MVM is the pseudo-inverse of the response matrix. The response matrix is generated by measuring the vector of WFS outputs produced by driving each single actuator of the deformable mirror. In our LCAOS, the response of any one pixel on the LCWFC cannot be observed by the WFS because the size of a pixel on the LCWFC is too small for a microlen on the WFS. Therefore the current wavefront reconstruction method is not suitable for our LCAOS.

A Zernike mode wavefront produced by the LCWFC can be observed by the WFS so we can first reconstruct the Zernike coefficients by the slopes on the WFS, and then compute the gray map by the Zernike coefficients. The process can be described by two MVM. The matrix of first MVM is called the reconstruction matrix, which is the pseudo-inverse of the response matrix. The response matrix in our LCAOS is generated by measuring the vector of WFS outputs produced by driving each Zernike mode wavefront on the LCWFC. The matrix of second MVM is called the Zernike matrix, which is calculated directly through Zernike polynomials.

The real-time processing of our LCAOS is shown in Fig. 4. The high speed camera sends the $100\times4\times4$ pixel raw sub-aperture

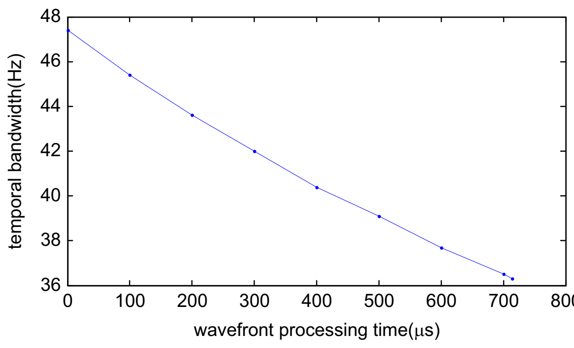

Fig. 3. Simulated temporal bandwidth of our LCAOS as a function of the wavefront processing time.

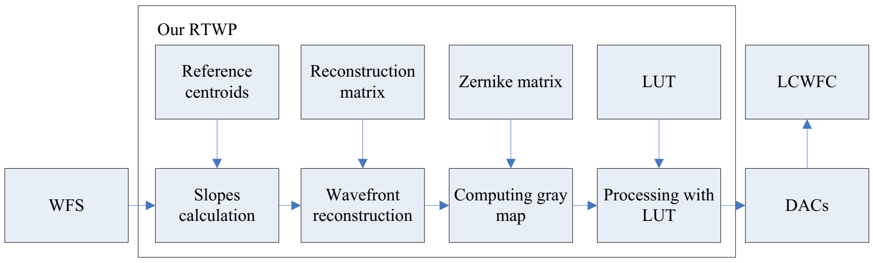

Fig. 4. Real-time processing for our LCAOS.

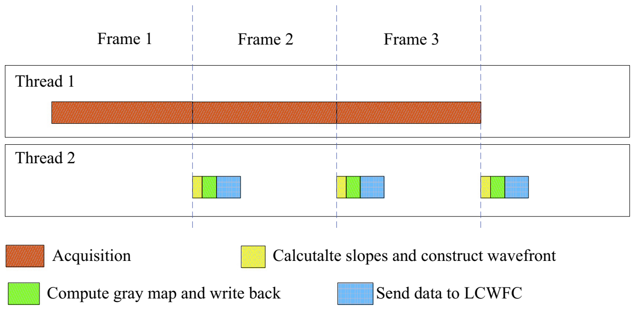
Frame 1
Frame 2
Frame 3
Thread 1
Thread 2
Acquisition
Calcutalte slopes and construct wavefront
Compute gray map and write back
Send data to LCWFC

Fig. 5. Schematic diagram of thread allocation for the main tasks.

images to the processor system. The computing slopes step determines the slopes from centroids shifts in the x and y directions of each sub-aperture, as compared to the reference centroids. The wavefront reconstruction step consists of a 209×200 reconstruction matrix, which is multiplied with the x and y slopes. The Zernike coefficient c is given as modal wavefront. The computing gray map step consists of 65536×209 Zernike matrix, which is multiplied with the Zernike coefficient. The gray map has been computed and is then sent to the DACs after applying look up table (LUT) processing.

The main tasks for our RTWP are the WFS's acquisition, wavefront processing and sending data to the LCWFC. In order to reduce context switching and mutexes, these main tasks should be assigned to a thread. But other tasks cannot be assigned to the acquisition thread while the WFS's CCD starts continuous acquisition. So the acquisition task monopolized a thread, and wavefront processing and sending data to the LCWFC occupied another thread. As shown in Fig. 5, thread 1 was responsible for the WFS's acquisition task; thread 2 was responsible for processing wavefront signal (including slopes calculation, wavefront reconstruction, computing gray map and data write back to the host) and sending data to the DACs. Except the essential main tasks, some necessary subtasks, such as parameter control, configuration and sharing of the real-time system information and diagnostic streams, also were assigned to different threads.

In our RTWP, a GPU card with GeForce GTX 590 from NVIDIA Corporation is used to construct wavefront and calculate the gray map so as to acquire powerful processing ability. Its technical specifications include two GPUs, each GPU has 16 multiprocessors and each multiprocessors has 32 stream processors with clock of 1215 MHz, so GTX 590 has 1024 stream processors, and a peak theoretical computing power is 2488.3GFLOPS; the memory clock is 1707 MHz and each GPU memory interface width is 384 bit, so peak theoretical memory bandwidth is 327.7 GB/s. In our wavefront processing, the performance is limited by GPU memory bandwidth rather than computing power, since memory data is read only once rather than repeatedly. Therefore, our work focuses on how to improve GPU memory bandwidth.

Before CUDA was released, general purpose GPU programming used graphics development kit. This requires programmers to package the data into a texture, the computing tasks mapped to the texture rendering process and use assembler or high-level shader programming language, such as GLSL, Cg, and HLSL. Programmers not only need to achieve the parallel algorithms, but also need to understand graphics programming. After CUDA was released, programmer can achieve parallel algorithms through C-like language without the need for any graphics programming. In CUDA, each stream processor can execute the same instruction on different data making it similar to a Single Instruction Multiple Thread (SIMT) processor. When a GPU program named as kernel can be executed by multiple equally-sharp blocks, and a block may contain up to 1024 threads, the total number of threads is equal to the number of threads per block times the number of blocks. Each block is a set of threads that can cooperate with one another by sharing data though some shared memory, but two threads from two different blocks cannot share data.

Kernel call on the GPU is a relatively expensive operation so the GPU is not suitable for small amount of calculation, such as the slopes calculation. We measured the time to call an empty kernel on the GTX 590 and the overhead of the kernel call is 10 μs. In contrast the slopes calculation on the CPU takes only 6 μs. Therefore it is necessary to assign slopes calculation task to the CPU. Since the number of Zernike model is 209 and the LCWFC has 256×256 pixels, the calculation of wavefront reconstruction and the gray map on the CPU requires a longer time. Therefore, to overcome the computation time problem, the wavefront reconstruct task and computing the gray map task are assigned to the GPU. The algorithm which can be executed on the GPU is described in the following sections.

4. Parallel computing algorithm with GPU

4.1. *Wavefront reconstruction*

Wavefront reconstruction is used to calculate 209 Zernike coefficients by a MVM operation. Only the threads in the same block can share data, so a Zernike coefficient must be calculated in the same block. Therefore the kernel for wavefront reconstruction is assigned 209 blocks and each block is assigned 200 multiply–add operation tasks. In order to execute the multiply–add operations on each block, the Mark Harris' method of parallel reduction is used [12].

Another method is to use a standard CUDA basic linear algebra subroutines (CUBLAS) library by NVIDIA Corporation. CUBLAS is a GPU-accelerated version of the complete standard basic linear algebra subroutines (BLAS) library, its cublasSgemv function perform single precision float point matrix–vector multiplication.

We use two different methods to measure the wavefront reconstruction on GTX 590. As shown in Table 1, the CUBLAS method delay is 22 μs, while the reduction method delay is 27 μs. Both methods are very low computing power for the GTX 590. This is because the amount of the wavefront reconstruction is too small, so that the computing power of the GTX 590 cannot be fully utilized. Nevertheless, the two methods meet our requirements. By contrast the CUBLAS method provides a speedup of 1.22, and is selected for our RTWP.

4.2. Computing gray map

Computing gray map is used to calculate 65536 gray values on the LCWFC. The MVM operation used for computing gray map is the computational bottleneck. The performance for computing gray map is limited by GPU memory bandwidth rather than computing power. The CUBLAS can reach about 83.6% of peak theoretical performance. Correspondingly we are able to reach about 85.1% of peak theoretical performance by the Vasily Volkov's optimization method: more registers in each thread [13] and unrolling parallel loops [14].

It is difficult to further improve performance for the standard MVM. But in the gray map calculation, if the Zernike matrix can be effectively simplified, we can reduce the data transmission time and obtain a lower computing latency.

The Zernike matrix consists of 209 Zernike map column vector, in other words, a vector consists of the driving voltage of 256×256 LCWFC in a unit corresponding Zernike polynomial. The Zernike polynomials are a set of functions that are orthogonal over the unit circle [15–17]. That is defined as

$$Z_i(r,\ \theta) = \begin{cases} N_n^m R_n^{|m|}(r)\cos(m\theta), & m \ge 0 \\ -N_n^m R_n^{|m|}(r)\sin(m\theta), & m < 0 \end{cases} \tag{4}$$

where

$$R_n^{|m|}(r) = \sum_{s=0}^{(n-|m|)/2} \frac{(-1)^s (n-s)!}{s![(n+|m|)/2-s]![(n-|m|)/2-s]!} r^{n-2s} \tag{5}$$

where $0 \le r \le 1$, N_n^m is the normalization factor and $0 \le \theta \le 2\pi$; n is non-negative integer and m varies from $-n$ to n with a step of 2; and i is ith Zernike polynomial.

In order to analyze the symmetry of the Zernike polynomial, we suppose that there is a point $P_1(\rho,\theta)$ $0<\rho<1,\ 0<\theta<\pi/2$. The points $P_2(\rho,\ \pi-\theta)$, $P_3(\rho,\ \pi+\theta)$, $P_4(\rho,\ 2\pi-\theta)$ are the symmetrical points of $P_1(\rho,\ \theta)$. The values of the Zernike polynomial at the symmetrical points can then be obtained by adding a negative sign or not to the value of the Zernike polynomial at P_1. Here, the symmetrical relation is classified by the parameter m as shown in Table 2.

To make it easy to understand the symmetrical relation of Zernike polynomial clearly, the symmetrical points are extended to the symmetrical regions. As shown in Fig. 6, relationship between the symmetrical regions is classified by parameter m. Only the first quadrant colored gray in each circle of Fig. 6 is the region that is actually computed. The values of the Zernike polynomial in the other regions are obtained by flipping the value of that in the first quadrant. For example, in Fig. 6(a), the first quadrant labeled S1 are the regions to be computed by the definition of the Zernike polynomial. The values of the Zernike polynomial in the other regions are obtained by flipping the value of the Zernike polynomial in the gray regions with respect to the y-axis and the x-axis. If there is a minus sign, the values will be multiplied by -1 after they are flipped.

In summary, once a quadrant of the Zernike polynomial is computed, the remaining regions of the unit circle can be simply obtained by flipping them. According to this conclusion, a complete Zernike map for the LCWFC can be obtained by flipping the first quadrant Zernike map and some examples are shown in Fig. 7. It is obvious that the Zernike map can be simplified to a quarter as shown in Fig. 7. The Zernike matrix is composed of 209 Zernike maps, so it can be effectively simplified to a quarter.

In conventional methods, the complete Zernike matrix and the Zernike coefficient vectors are read to GPU, and then the MVM operation is executed in GPU. In our Zernike symmetry method, first the simplified Zernike matrix and the Zernike coefficient vector are read to GPU; then the complete Zernike matrix is restored by flipping the simplified Zernike matrix in GPU; finally, the MVM operation is executed in GPU. The size of the complete Zernike matrix is 54.8 MB, and then the size of the simplified Zernike matrix is reduced to 13.7 MB. Because the computational bottleneck is the read bandwidth from GPU memory to GPU, the computational time can be reduced to 1/4 by using our Zernike symmetry method in theory. In fact, the actual result cannot reach

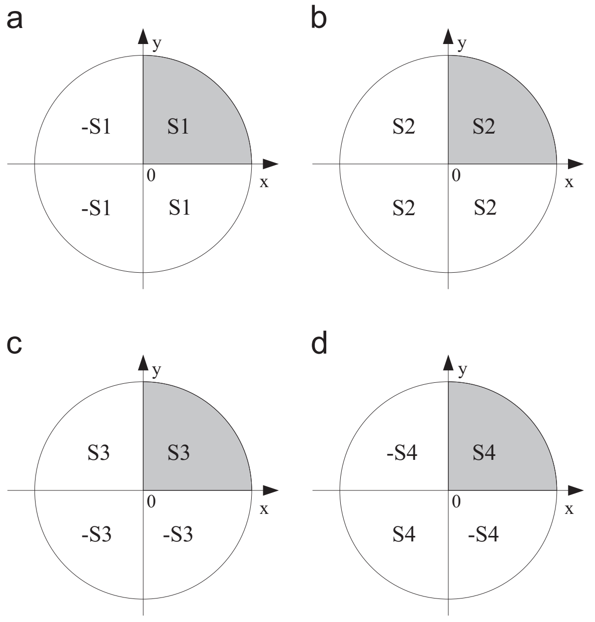

Fig. 6. Relationship between the symmetrical regions, (a) $m \ge 0$ and m is odd, (b) $m \ge 0$ and m is even, (c) $m<0$ and m is odd, (d) $m<0$ and m is even.

Table 1
Results of wavefront reconstruction for different methods.

	Reduction method	CUBLAS library
Computing latency	27 μs	22 μs
Computing power	3.1 GFLOPS	3.8 GFLOPS

Table 2
Relationship between the symmetrical points.

Symmetrical point	Value at symmetric point	$m \ge 0$		$m < 0$	
		m is odd	m is even	m is odd	m is even
P_1	$Z_i\ (r,\ \theta)$	$Z_i(r,\ \theta)$	$Z_i\ (r,\ \theta)$	$Z_i\ (r,\ \theta)$	$Z_i\ (r,\ \theta)$
P_2	$Z_i\ (r,\ \pi-\theta)$	$-Z_i\ (r,\ \theta)$	$Z_i\ (r,\ \theta)$	$Z_i\ (r,\ \theta)$	$-Z_i\ (r,\ \theta)$
P_3	$Z_i\ (r,\ \pi+\theta)$	$-Z_i\ (r,\ \theta)$	$Z_i\ (r,\ \theta)$	$-Z_i\ (r,\ \theta)$	$Z_i\ (r,\ \theta)$
P_4	$Z_i\ (r,\ 2\pi-\theta)$	$Z_i\ (r,\ \theta)$	$Z_i\ (r,\ \theta)$	$-Z_i\ (r,\ \theta)$	$-Z_i\ (r,\ \theta)$

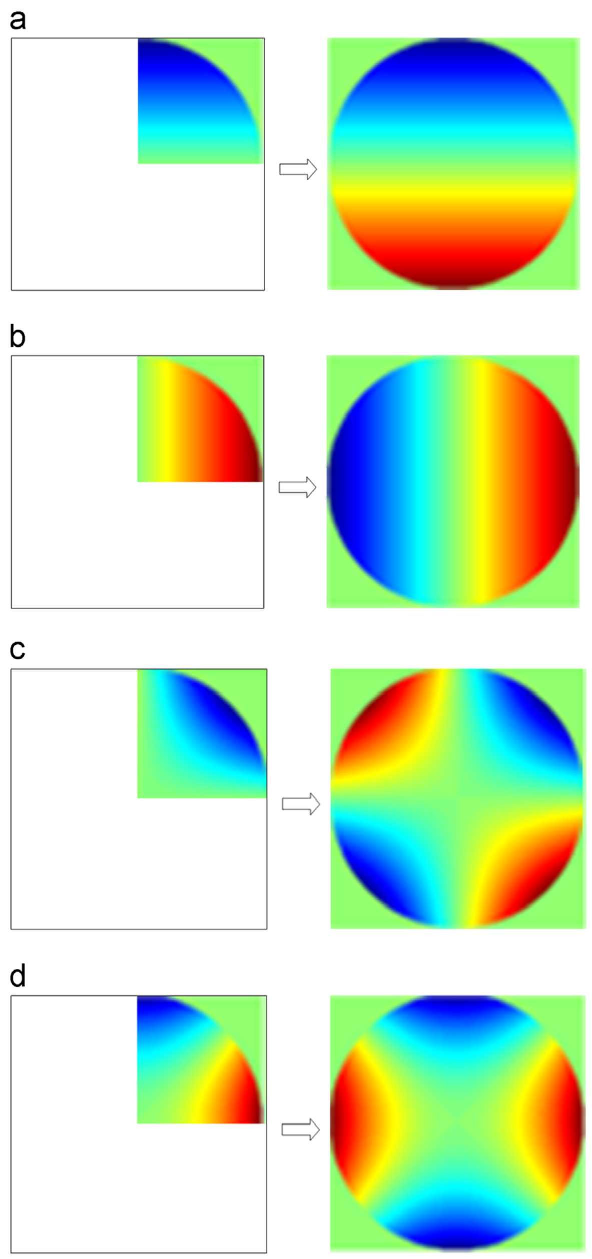

Fig. 7. Four types of Zernike maps obtained by flipping the Zernike map on the first quadrant. (a) mode 1 and $m=-1$ ($m<0$ and m is odd), (b) mode 2 and $m=1$ ($m>0$ and m is odd), (c) mode 3 and $m=-2$ ($m<0$ and m is even), (d) mode 5 and $m=2$($m>0$ and m is even).

the theoretically expectation, because transmission speed is not fixed: the smaller the amount of data, the slower transmission speed. Anyway, our Zernike symmetry method can effectively reduce the computing latency and improve performance. The results of computing gray map for different methods are shown in Table 3.

As shown in Table 3, the computing latency is reduced from 196 μs to 62 μs and our Zernike symmetry method provide a speedup of 3.16, although the effective read memory bandwidth is reduced from 279 GB/s to 221 GB/s.

By the way, as another common modal functions in AO, Karhunen–Loève (K–L) functions can replace Zernike polynomials for our wavefront processing algorithm[18]. Because K–L functions and Zernike polynomials have the same symmetry, our symmetry method on GPU also can be used while we used K-L functions as our wavefront processing algorithm.

Table 3
Results of computing gray map for different methods.

	CUBLAS library	Our original method	Our Zernike symmetry method
Computing latency	200 μs	196 μs	62 μs
Computing power	137 GFLOPS	140 GFLOPS	442 GFLOPS
Read memory size	54.8 MB	54.8 MB	13.7 MB
Read memory bandwidth	274 GB/s	279 GB/s	221 GB/s
Read bandwidth percentage	83.6%	85.1%	67.4%

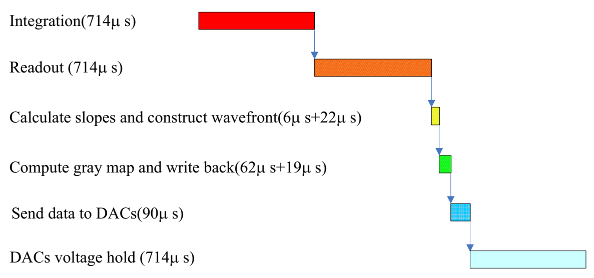

Fig. 8. Latency of the total computing process.

4.3. *Processing with LUT*

The computed gray map is further processed by look-up table (LUT) before sending to the LCWFC. To save time, the LUT is also conducted in the GPU. The texture cache in CUDA is optimized for LUT correction, so that we read the LUT through texture cache. The LUT processing does not take up more time, because its time is hidden in data transmission for computing gray map. Finally, there is an additional latency for computing gray map as data needs to be written-back from GPU memory to the host pinned memory. The pinned memory is stored in the physical memory, which enables a direct memory access (DMA) on the GPU to request transfers to the host memory without the involvement to the CPU.

4.4. *Combining MVM*

Wavefront reconstruction and computing gray map are both MVM. In mathematics, they can be combined into a MVM and the size of the combined matrix is 65536×200. The combined MVM does not have symmetry, so its computing latency is measured to be 190 μs. By contrast, the computing latency of wavefront reconstruction and computing gray map are only 22 μs and 62 μs. Therefore the combined MVM cannot provide a lower computing latency in our LCAOS.

5. Results

Our RTWP consisted of an Intel Xeon x5687 4-core CPU clocked at 3.60 GHz and a GeForce GTX 590 GPU card. We used CUDA 4.2 release for implementing on the GPU card. A real-time Linux kernel (with the RT-preempt patch, 2.6.29.6-rt24, available from CentOS archives) was used. The performance of our RTWP was assessed by using the Linux real-time clock to measure computing time. Fig. 8. shows the processing timing line on our RTWP. As shown, the total wavefront processing time was measured to be 109 μs. The wavefront processing jitter of 5 μs rms was measured, averaged over 1 million consecutive frames.

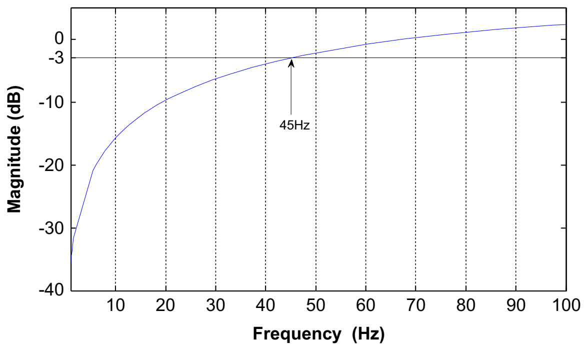

Fig. 9. Magnitude of the residual error transfer function in our LCAOS.

A common Linux kernel (2.6.29.6) was also used. It provides a nearly identical performance: the wavefront processing time of 109 μs and the wavefront processing jitter of 6 μs rms. Therefore, a real-time Linux kernel is not necessary in our LCAOS.

Once the wavefront processing time is determined, the magnitude of the residual error transfer function $\varepsilon(s)$ can be simulated, which is shown in Fig. 9. As shown, the temporal bandwidth $f_{-3\ \mathrm{dB}}$ of our LCAOS can be up to 45 Hz.

6. Conclusions

According to our previous study, the residual wavefront error is mainly caused by temporal error [19]. The system temporal error can be calculated by

$$\sigma_{temp}{}^2 = \left(\frac{f_G}{f_{-3\ \mathrm{dB}}}\right)^{5/3} \tag{6}$$

where f_G is the Greenwood frequency, and $f_{-3\ \mathrm{dB}}$ is the temporal bandwidth of our LCAOS. In our astronomical observation area, the Greenwood frequency f_G is usually around 45 Hz. The temporal bandwidth $f_{-3\ \mathrm{dB}}$ of our LCAOS is 45 Hz, so that the temporal error is calculated to be 1 rad^2 by Eq. (6). In order to improve the temporal bandwidth, we need to improve the response speed of the liquid crystal material to reduce decay time τ_{decay} and choose a higher frequency of the detector to reduce integration time T of the WFS.

In summary, we present a cost-effective scalable real-time wavefront processor. We have described some algorithms for wavefront reconstruction, computing gray map and LUT process based on GPU. Especially, the algorithm by using symmetry of Zernike polynomial can effectively reduce the calculation latency of gray map to less than half and provide a speedup of 3.16. Finally, the wavefront processing time can be achieved in 109 μs. Correspondingly, the system temporal bandwidth can be up to 45 Hz.

The hardware architecture of our RTWP is based on commercial off-the-shelf CPU and GPU. It can easily get a performance upgrade through the replacement of hardware at a low cost in future. The compiled program can be easily ported to upgrade hardware. With higher performance and additional GPUs, we expect that the architecture will easily support our next generation LCAOS at higher rates and lower latency.

Acknowledgments

This work is supported by the National Natural Science Foundation of China with Grant numbers of 11174274, 11174279, 61205021, 11204299, 60736042, 60578035 and 50703039.

References

[1] G.D. Love, Appl. Opt. 36 (7) (1997) 1517.
[2] D. Dayton, J. Gonglewski, S. Restaino, J. Martin, J Phillips, M. Hartmann, P. Kervin, J. Snodgress, S. Browne, N. Heimann, M. Shilko, R. Pohle, B. Carrion, C. Smith, D. Thiel, Opt. Express 10 (25) (2002) 1508.
[3] Q. Mu, Z. Cao, L. Hu, D. Li, L. Xuan, Opt Express 14 (18) (2006) 8013.
[4] Z. Cao, Q. Mu, L. Hu, D. Li, L. Xuan, Acta Photonica Sin. 37 (2008) 785.
[5] C. Liu, L. Hu, Q. Mu, Z. Cao, L. Xuan, Appl. Opt. 50 (1) (2011) 82.
[6] Z. Cao, Q. Mu, L. Hu, D. Li, Y. Liu, L. Jin, L Xuan, Opt. Express 16 (10) (2008) 7006.
[7] L. Hu, L. Xuan, D. Li, Z. Cao, Q. Mu, Y. Liu, Z. Peng, X. Lu, Opt. Express 17 (9) (2009) 7259.
[8] J.G. Marichal-Hernandez, L.F. Rodriguez-Ramos, F. Rosa, J.M. Rodriguez-Ramos, Appl. Opt. 44 (35) (2005) 7587.
[9] T.N. Truong, A.H. Bouchez, R.G. Dekany, J.C. Shelton, M. Troy, J.R. Angione, R.S. Burruss, J.L. Cromer, S.R. Guiwits, J.E. Roberts, SPIE 7015 (2008) 31.
[10] A.G. Basden, R.M. Myers, Mon. Not. R. Astron. Soc. 424 (2012) 1483.
[11] NVidia, CUDA C Programming Guide, ⟨http://docs.nvidia.com/cuda/index.html⟩.
[12] M. Harris, Optimizing parallel reduction in cuda, ⟨http://developer.download.nvidia.com/asserts/cuda/files/reduction.pdf⟩.
[13] Volkov, Unrolling parallel loops, tutorial talk at SC11. 2011.
[14] Volkov, Use registers and multiple outputs per thread on GPU, International Workshop on Parallel Matrix Algorithms and Applications 2010 (PMAA'10), 2010.
[15] R.J Noll, J. Opt. Soc. Am. 66 (3) (1976) 207.
[16] Z. Cao, Q. Mu, L. Hu, X. Lu, L. Xuan, Opt. Express 17 (20) (2009) 17715.
[17] Q. Mu, Z. Cao, Z. Peng, Y. Liu, L. Hu, X. Lu, L. Xuan, Opt. Express 17 (11) (2009) 9330.
[18] G. Dai, J. Opt. Soc. Am. 13 (6) (1996) 1218.
[19] Z. Cao, Q. Mu, L. Hu, Y. Liu, Z. Peng, Q. Yang, H. Meng, L. Yao, L. Xuan., Opt. Express 20 (17) (2012) 19331.