J Real-Time Image Proc
DOI 10.1007/s11554-017-0702-7

CrossMark

ORIGINAL RESEARCH PAPER

Suitability of GPUs for real-time control of large astronomical adaptive optics instruments

Urban Bitenc[1] · Alastair G. Basden[1] · Nigel A. Dipper[1] · Richard M. Myers[1]

Received: 8 August 2016 / Accepted: 16 June 2017
© The Author(s) 2017. This article is an open access publication

Abstract Adaptive optics (AO) is a technique for correcting aberrations introduced when light propagates through a medium, for example, the light from stars propagating through the turbulent atmosphere. The components of an AO instrument are: (1) a camera to record the aberrations, (2) a corrective mechanism to correct them, (3) a real-time controller (RTC) that processes the camera images and steers the corrective mechanism on milliseconds timescales. We have accelerated the image processing for the AO RTC with the use of graphics processing units (GPUs). It is crucial that the image is processed before the atmospheric turbulence has changed, i.e., in one or two milliseconds. The main task is to transfer the images to the GPU memory with a minimum delay. The key result of this paper is a demonstration that this can be done fast enough using commercial frame grabbers and standard CUDA tools. Our benchmarking image consists of 1.6×10^6 pixels out of which 1.2×10^6 are used in processing. The images are characterized and reduced into a set of 9248 numbers; about one-third of the total processing time is spent on this characterization. This set of numbers is then used to calculate the commands for the corrective system, which takes about two-third of the total time. The processing rate achieved on a single GPU is about 700 frames per second (fps). This increases to 1100 fps (1565 fps) if we use two (four) GPUs. The variation in processing time (jitter) has a root-mean-square value of 20–30 µs and about one outlier in a million cycles.

✉ Urban Bitenc
urban.bitenc@durham.ac.uk

1 Centre for Advanced Instrumentation, Durham University, Durham, UK

Keywords Adaptive optics (AO) · Extremely large telescope (ELT) · Real-time control (RTC) · Graphics processing unit (GPU)

1 Introduction

Adaptive optics (AO, [1]) is used to correct the aberrations introduced when light propagates through a medium. In astronomical observations, AO compensates the distortions caused by the atmospheric turbulence [2]. The key parts of an AO instrument are a camera that records the distortions of light, and a real-time controller (RTC) that processes the camera images and steers the mechanism correcting the distortions.

Extremely large telescopes (ELTs, e.g., [3]) will crucially depend on AO; without the AO, the atmospheric aberrations will void any improvement in the resolution due to the larger telescope diameter. While AO is now well established on 8 and 10 meter class telescopes, its extension to 30–40 m telescopes remains a significant challenge. One major aspect of that challenge is the provision of a suitable low-latency RTC.

Since the release by NVidia of the CUDA development environment, GPUs have been a popular technology for the acceleration of AO systems [4, 5]. For this paper, we have implemented the algorithms for AO real-time control on graphics processing units (GPUs) within DARC (the Durham AO real-time controller [7]), and we studied its performance for an ELT-size system. One of the key tasks was minimizing the delay due to the copying of camera images from the CPU memory to the GPU memory. Sevin et al. [8] have developed a way to copy the camera images directly from the camera into the GPU memory, without involving the CPU memory. While this will clearly provide a better performance, it is based on custom developed hardware and software. The alternative solution, which we fully investigate in this paper, is to use a commercial frame grabber to copy the camera data into the CPU memory and then use the standard CUDA tools to copy the data from the CPU memory into the GPU memory. We demonstrate that the performance of this non-custom solution still has the potential to satisfy the requirements of ELT instruments.

The Xeon Phi, which can be used in a conceptually similar way as the GPU, was found to exhibit a larger amount of jitter [5, 6]. However, that may improve with newer versions.

An alternative approach to acceleration of the RTC is to develop faster algorithms; these can provide a twofold to threefold improvement in processing time. Several such algorithms have been proposed and tested on-sky, including the Fourier transform reconstructor [10] and CuReD algorithm [11]. While this algorithmic approach is valuable and will be required for the highest order AO systems, the conventional algorithm is much more widely tested. As we will show, it will be possible to use the conventional algorithm on at least some of the ELT instruments by using GPUs.

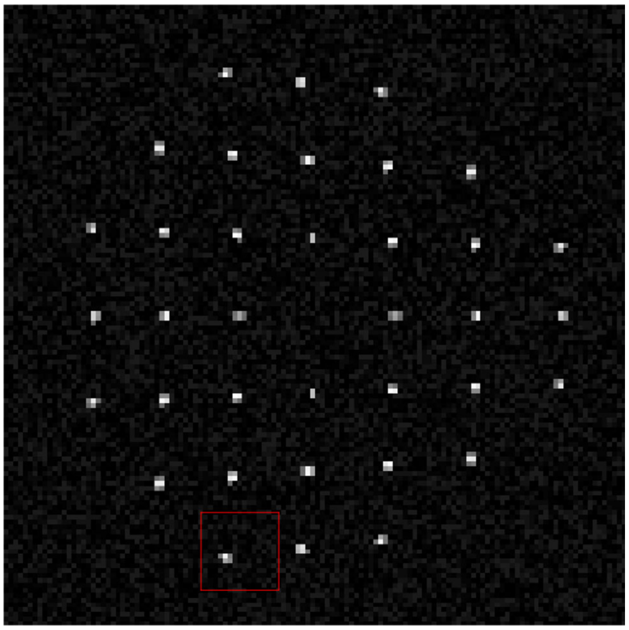


Fig. 1 A typical image of a wavefront sensor camera on an AO system with 7×7 subapertures (from computer simulation). The borders of one of the subapertures are shown in *red*. The central spot is missing because it is *shaded* by the telescope's secondary mirror

2 Brief description of adaptive optics

AO systems are composed of three subsystems. The wavefront sensor (WFS) produces an image of the aberrations, the real-time controller (RTC) processes this image and computes the optimal correction commands, and the corrective subsystem corrects the aberrations. Read-out of the WFS camera, the image processing and the application of the commands on the corrective system must happen rapidly, before the atmosphere has changed significantly. Due to atmospheric coherence time, this is typically within a millisecond [2, 5, 6].

The wavefront is defined as the surface of constant phase of the electromagnetic field, and it is perpendicular to the direction of propagation of the light. A perfectly flat wavefront corresponds to non-aberrated light; aberrations can be measured by detecting the deviation of the wavefront from the perfectly flat shape. A Shack–Hartmann WFS uses a lenslet array to measure the local derivatives of the wavefront on a grid of points. The lenslets generate light spots in the image plane of the camera, as shown in Fig. 1. The position of each spot directly relates to the wavefront derivative in the corresponding region. A group of camera pixels corresponding to a lenslet is called a subaperture, i.e., a subarea of the telescope aperture.

The RTC processes the images and extracts the wavefront derivatives. Then, it uses the derivatives to calculate the steering commands for the corrective part of the AO system. The latter is usually a deformable mirror the shape of which is modified in real-time to cancel the aberrations introduced by the atmosphere.

2.1 Image processing in adaptive optics

The AO image processing is performed in three steps: (1) image calibration, (2) calculation of the wavefront derivatives and (3) calculation of the steering commands.

The image calibration is performed in three operations: each pixel of the image is multiplied by the calibration factor; then, the value of the background is subtracted and finally; if the resulting value is below the threshold, the value is set to 0.0 to reduce the effect of noise. The telescope's primary mirror has a circular geometry, whereas the WFS camera has a square one. The "corners" of the WFS image are not illuminated and do not contain any signal, see Fig. 1. To speed up the calibration, we excluded the non-illuminated regions, reducing the number of pixels to be calibrated from 1.6×10^6 to 1.2×10^6.

To calculate the wavefront derivatives, the position of each light spot on the image is calculated. In our case, light spots are spread over several pixels and hence the center-of-gravity method is used to calculate the x and y positions of the center of the light spot. From the values obtained, the nominal value, corresponding to a flat wavefront, is subtracted and the result is proportional to the local derivative of the wavefront.

The steering commands are calculated by multiplying the array of derivatives with a so-called control matrix (matrix-vector multiplication, MVM). For large AO instruments, this computation takes the majority of the image processing time; for our benchmark, it took about 70% of the total computation time (the other 30% are taken by the image calibration and by the calculation of derivatives).

3 Implementation on GPUs

We have implemented the AO image processing algorithm on GPUs, within the framework of DARC.

3.1 Durham AO real-time controller (DARC)

DARC [7] is a flexible, modular real-time control system for AO that is primarily CPU-based, but can have optional hardware acceleration modules. It was first developed for use with the CANARY on-sky AO demonstrator instrument [12] and has since seen use with several other instruments worldwide. DARC aims for computational efficiency using a horizontal processing strategy, where all processing threads perform similar tasks to optimize load balancing, rather than a more conventional strategy where some threads will perform calibration, some slope computation and some wavefront reconstruction. A horizontal strategy leads to a significant reduction in inter-thread communication requirements. The achieved measured performance of DARC makes it suitable for ELT use with appropriate computational hardware. Key flexibility is provided by the modular design, with dynamic loading and unloading of modules allowing development and testing of new algorithms without a system restart. This therefore makes DARC highly suited to operation in laboratory environments, where continued system development is often necessary.

3.2 Details of the implementation

We set up DARC in such a way that the wavefront sensor camera frame is split into a number of chunks (groups of subapertures). The number of these chunks is a key parameter of the system and is denoted by N_C. Each chunk is assigned to its own CPU thread for processing, and each thread is coupled to a CUDA stream which controls copying the chunks of data to the GPU and processing them. If several GPUs are used, the chunks are distributed to the GPUs evenly, so that all GPUs perform a similar amount of computation. When all the streams on a GPU have completed processing and have produced their partial output, these partial outputs from all streams are summed up and copied back to the CPU. If more than one GPU is used, the CPU sums up the outputs from all GPUs. The final output is then virtually sent to a deformable mirror and the processing of the next camera image begins. (For our test, no actual deformable mirror is used.)

The processing pipeline consists of 12 steps as summarized in Table 1. These steps are:

1	Copy the pixels from the subapertures used to a separate buffer. The memory for this buffer was allocated using cudaMallocHost(), which results in the so-called pinned memory being used. This enables one to start processing one subaperture chunk (step 8) as soon as all its pixel data have been copied to the GPU (step 7). While this chunk is being processed, other chunks of data are being copied from CPU to GPU.
2, 3	Each CPU thread launches the command to copy the pixels to the GPU and launches the kernel calls into the CUDA stream corresponding to this thread.
4–6	One of the CPU threads launches a kernel to sum up the partial outputs from all the chunks processed on that GPU and launches the command to copy the output back to the CPU. These are launched into the default CUDA stream. For synchronization with the GPU, it creates an event “copied OK”.
7, 8	Each CUDA stream copies its data to the GPU and processes it.
9–11	When the last CUDA stream has finished processing its data, the default stream sums up all the partial outputs on this GPU and copies them to the CPU. Finally it records the event “copied OK” to signal the CPU that processing on that GPU has completed.
12	After the “copied OK” event has been recorded by a GPU, the CPU adds the output of this GPU to the final output array.

For pixel calibration and for the center-of-gravity calculation, we developed the GPU kernels ourselves. For the matrix-vector multiplication, we used the CUBLAS function cublasSgemv as a starting point and customized it for our particular use-case to improve performance.

For configurations using more than one GPU, we explored several options for the synchronization between the CPU and the GPUs at the end of each processing cycle. The optimal results were obtained using “cudaEventQuery()” which continuously polls all GPUs, checking whether any of them has finished processing.

Table 1 Steps of the image processing

	CPU Each thread	CPU One thread only	GPU Each stream	GPU One stream only
1	Copy pixels used to abuffer. Then launch:			
2	Copying pixels to GPU,			
3	Image processing.			
		Launch		
4		Sum up output on GPU,		
5		Copy output to CPU,		
6		Event “copied OK.”		
7			Copy pixels to GPU, process the image	
8				
9				Sum up output on GPU,
10				copy output to CPU,
11				record “copied OK”
12		Sum up output on CPU		

The CPU prepares pixel data, launches the data copy commands and the processing kernels and finalizes the output in the end. The GPU copies the data and processes it

3.3 Correlation wavefront Sensing

Cross-correlation is an optional addition to the calculation of wavefront derivatives to improve its performance in cases when the light spot has a bigger size [9]. This calculation consists of five steps: zero-padding each subaperture (from 16×16 to, e.g., 32×32), Fourier transform of each subaperture, complex multiplication with the reference, inverse Fourier transform and clipping the subaperture edge to speed up the center-of-gravity calculation. The result of this cross-correlation is then passed on to the centroiding algorithm.

We used the library “cufft” to calculate the Fourier transforms. Note that with correlation the number of kernel launches per chunk of subapertures increases from three to eight.

4 Benchmarking

The results presented here were obtained using the GPU devices K20Xm and K80. The error-correcting code mechanism was deactivated to investigate the maximum achievable performance, and the clock rate of K20Xm was increased from the default 732 MHz to its maximum value of 784 MHz.

4.1 System configuration

We set up DARC for a virtual single conjugate AO (SCAO, the simplest AO configuration, see [2]) system with a grid of 80×80 subapertures. The system parameters are given in Table 2.

For the majority of our tests, no physical camera was used. The camera images were read in from a file when starting the application.

4.2 Configuration of the GPU host computer

In our initial measurements of image processing time for each cycle, these times varied significantly between consecutive cycles, exhibiting a distribution with the root-mean-square of several 100 µs, strongly non-Gaussian shape and regular outliers of several tens of milliseconds. This phenomenon is called jitter and is due to other processes running on the CPU and due to dynamic scheduling (both on the CPU and on the GPU), resulting in a non-deterministic order of memory access and similar effects. For adaptive optics instruments, such behavior has to be

Table 2 AO system configuration used for benchmarking

AO system type	SCAO, one WFS
Subaperture grid	80×80
Number of subapertures used	4624
Subaperture size (in pixels)	16×16
Subap. size for correlation	32×32
Number of controlled actuators	4828
MVM size	9248×4828

The number of subapertures used is smaller than $80 \times 80 = 6400$ because the subapertures outside the circle covered by the telescope mirror are not used minimized; the repeatability of the processing time is of key importance for the quality of the correction of light aberrations. We took the following steps to minimize the jitter:

- Use the *lowlatency* Linux kernel (rather than *generic*).
- Switch off power-saving, i.e., set the CPU frequency scaling_governor to *performance* (rather than to *ondemand*).
- We set thread affinity so that each CPU thread is forced to run on exactly one hyper-thread. We investigated which hyper-threads work best for which GPU device. We set the thread priorities to 99.

With these settings, the jitter exhibits a close-to-Gaussian distribution with the root-mean-square of a few 10 µs (see Fig. 4; Table 3) and on average one or two outliers in one million cycles.

One expects that the lowest jitter will be obtained when using the real-time patch (PREEMPT_RT) for the Linux kernel. However, NVidia drivers are only supported for the *generic* and *lowlatency* kernels; hence, tests with the real-time patch were not possible. Our initial test showed that the *lowlatency* kernel gives less jitter than *generic*. Smith et al. [13] suggest that after applying all the other settings the *generic* kernel would perform similarly, but we have not verified that.

4.3 Optimizing the parameters

The goal is to achieve a high average frame-per-second rate (i.e., process images as quickly as possible) with a low variation of processing times for each frame (i.e., low jitter). Several parameters and options have to be tuned to achieve the optimal performance, the most important being:

- The number of subaperture chunks, N_C,
- which CPU threads run on which CPU cores.
- use or no-use of mutex_lock for each CPU thread when the thread is launching the kernels.
- the number of CUDA threads per block
  - in the pixel calibration kernel,
  - in the MVM kernel,
- the extent of loop unroll in the MVM kernel.

The optimal values of these parameters depend on the GPU type, on the number of GPUs used and on the architecture of the server hosting the GPUs.

The most important parameter is N_C. When N_C is increased, the GPU resources are utilized better for two reasons. First, more of the data is processed in parallel to other data being copied, and second, smaller data chunks can generally fill the available GPU resources better (in the same way as a number of small boxes fill the available space better than a few large boxes). However, each subaperture chunk requires a launch of three kernels and a CPU thread controlling it, adding some overhead for each additional subaperture chunk. Hence, there is an optimal N_C with an optimal trade-off between these effects.

Figure 2 shows an example of parameter optimization. The upper plot demonstrates that in case of one GPU, the criterion of high speed conflicts with the criterion of low jitter, so one needs to make a trade-off. The lower plot shows that if using two GPUs, the fastest solution will also have the lowest jitter, which is an unexpected and a very positive result. The latter trend is also observed when using 3 or 4 GPUs.

4.4 Results

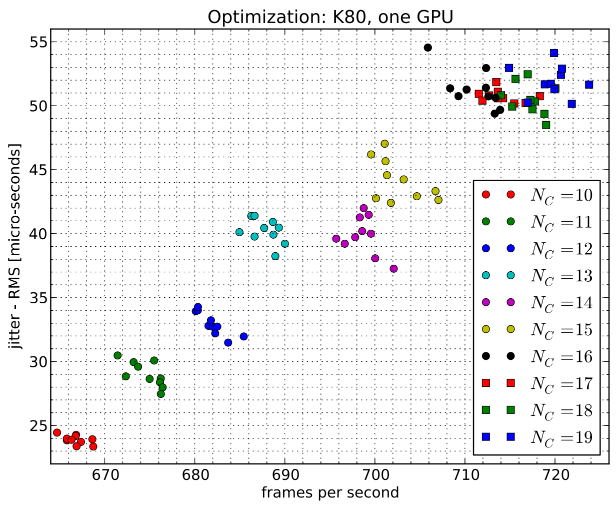
We benchmarked a number of NVidia GPUs: Quadro 600, GeForce GTX 580 and 780Ti, Tesla C2070, K20Xm, K40 and K80. It turned out to be non-trivial to understand the correlation between the GPU properties and the observed performance; the best figures of merit are the number of cores and clock speed, but not necessarily GPU's age or price. One of the fastest GPUs was GTX 580, which is older and cheaper than most of the GPUs we studied.

The best results were obtained with the K80 device that contains two GPUs. Using only one of the two GPUs, the frame rate achieved is 670 frames per second (fps); this number can be increased up to 720 fps but then the jitter also increases, see the upper plot in Fig. 2. Using both GPUs of the K80 device, the rate achieved is 1100 fps. By deploying additional two GPUs K20Xm hosted in the same server, the rate increases to 1565 fps. With the error-correcting code activated (which guarantees the absolute correctness of the result), the achieved frame rates were about 10% lower. The frame rates achieved are given in

Table 3 Frame rates and jitter achieved by different configurations

Configuration	N_C	Average frame rate (fps)	RMS of jitter (µs)	Minimum frame rate (fps)
K80, one GPU	10	672	23	620
K80, two GPUs	18	1111	27	970
K80 + 2 × K20	28	1565	32	1170

The right-most column gives the frame rate with which 99.999% of cycles will complete processing before the next camera frame is ready



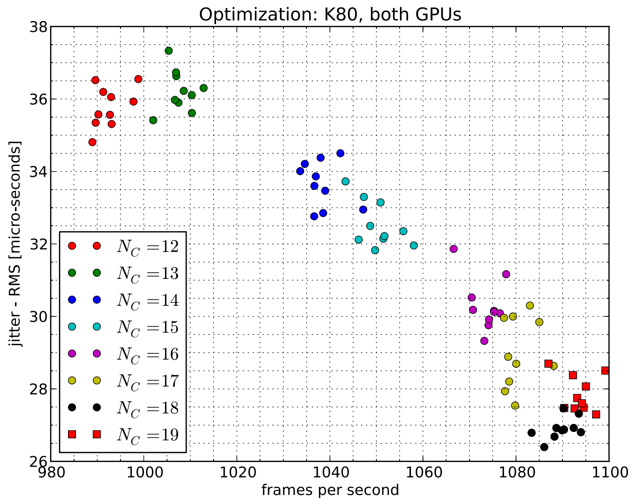


Fig. 2 Illustration of parameter tuning. *Different markers* represent different values of N_C. Different points for the *same marker* show the values obtained with different loop unroll size in the MVM kernel. The *upper* plot shows the results when using one of the two GPUs from K80, and the *lower* plot when using both GPUs

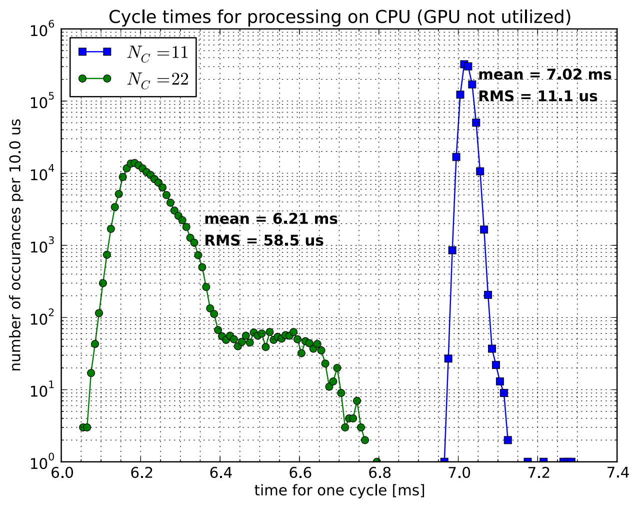


Fig. 3 Distribution of processing times if GPUs are not utilized and all processing is done on the CPU. Two typical configurations are shown: the faster one is wider (more jitter) and the slower one is narrower (less jitter)

Table 3 and are shown in Fig. 4. These results have to be compared to the rates required for different ELT instruments, which vary from 250 to 1000 fps or even 2000 fps.

We performed further test, showing that the K20Xm GPUs are about 10% slower than the K80; hence, we conclude that with two K80 devices the frame rate would probably exceed 1600 fps. Using more than four GPUs would probably provide little increase in frame rate, for the reasons discussed in Sect. 5.2.

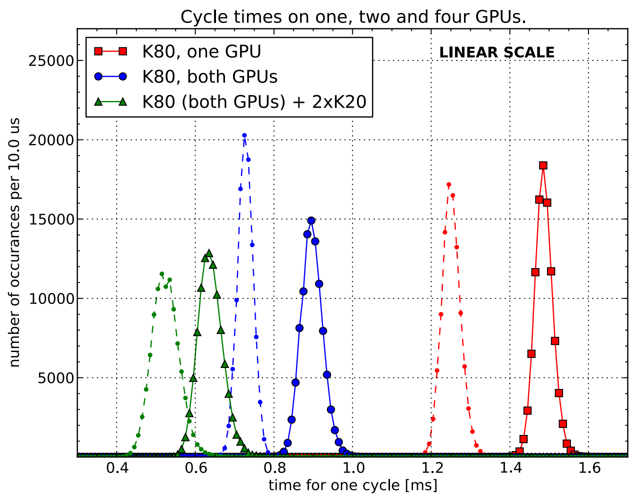
Performing the same test on the CPU, we achieved rates of up to 160 fps, see Fig. 3. The use of two K80 devices in this case provides a tenfold increase in speed.

The spread of cycle times obtained with the GPUs is relatively low and has an RMS of 20–30 µs. A further investigation shows that about half of the jitter comes from the data processing on GPU and about a quarter from copying the camera images between the CPU to the GPU. For comparison, the RMS of the jitter obtained by the CPU is 11 µs. When using four GPUs instead of one, the jitter increases by about 50% only, which is an unexpectedly positive result.

Note that in Fig. 4 we only show distributions for 100,000 cycles as afterward the temperature of the K80 device, if both GPUs are used, increases to a point where the GPU's clock rate is reduced automatically. To characterize the outliers, we have performed several longer tests with one GPU only and with the two K20Xm GPUs. We typically observe one or two outliers in one million cycles. The largest outlier we observed in over 30 million cycles was at 4.8 ms.

These results clearly demonstrate that by using one or two GPU devices, an 80×80 system can be comfortably controlled with rates well above 1 kHz. We achieved this by using only commercial off-the-shelf components for transferring the camera images from the WFS camera to the GPU via the CPU memory.

The results obtained deploying the correlation wavefront sensing are given in Table 4. The optimal N_C is lower than in Table 3 which is due to a higher number of kernel launches (eight instead of three). We have not fully explored the available parameter space to find the fastest configuration, and also further optimizations of the code may be possible. Nevertheless, these results demonstrate the capability to control an 80×80 system with the cross-correlation algorithm, with a rate higher than 500 Hz.



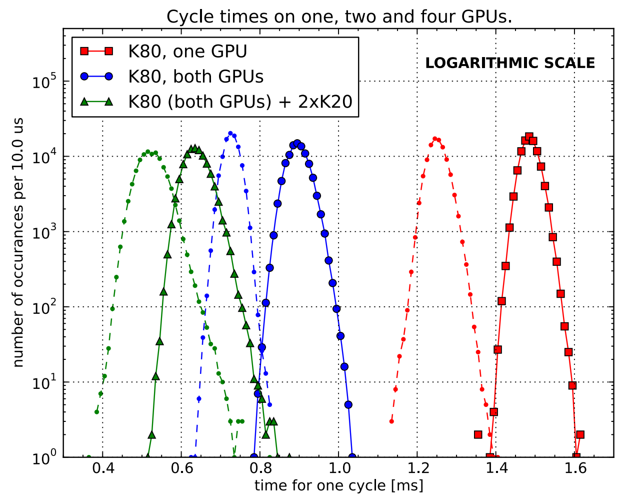


Fig. 4 Distributions of processing times for different configurations, shown in linear (*upper*) and logarithmic scale (*lower*). The *red*, *blue* and *green lines* show the cycle times achieved by using one, two and four GPUs. The full lines are for the complete process, whereas the *dashed lines* show the times achieved if the camera images are not copied from CPU to the GPU

4.5 Test with a real camera

For the majority of our tests, a camera image was read in from a file when starting the application and then the same image was used on every iteration.

However, we also performed a test with real data from a 10G Ethernet camera, EVT HS-2000. This camera has 1088×2048 pixels and can run at 338 Hz, delivering full frames. For this test, we used a different GPU, GeForce 580. The frame rate obtained with the pixel input from the real camera was 510 fps which is very close to the values obtained by using the images from a file. (The subapertures were reshaped from 16×16 pixels to 8×25 pixels to enable the camera to deliver frames at this rate. The GPU processing time is only minimally affected by this decreased subaperture size.) The jitter was also similar showing that our results are equally applicable to real-camera data.

5 Lessons learned

The main conclusions of our work are the following.

5.1 Copying camera images to GPU

One of the main objectives of this study was to investigate the impact of copying the camera images from the CPU memory to the GPU memory.

The step of copying the camera images to GPU takes about 1 ms (or 0.5 ms with PCIe 3.0) on its own; this includes the extraction of only the pixels that are used (Table 1, step 1). This would directly increase the time delay before the AO correction can be applied to correct for the atmospheric distortion. However, by processing the data in parallel to copying them, the overhead of copying is reduced by 80–90%.

To demonstrate the potential gain of transporting the camera images from the camera directly to the GPU, we perform the following test. We skip the step of copying the camera images from the CPU to the GPU; the GPUs are then processing pixel data which are all 0.0. The average frame rate increases by about 10–15% if using one GPU, and by about 20–25% if using two or four GPUs. The jitter increases by about 20–50%. The corresponding distributions are shown with dashed lines in Fig. 4.

The solution which copies the camera images directly into the GPU memory would obviously perform better. However, we conclude that the solution transporting the data via the CPU memory, using standard tools, is not much worse and presents a good candidate for the RTC hardware for ELT instruments.

Table 4 Frame rates and jitter achieved when the cross-correlation algorithm is activated

Configuration	N_C	Average frame rate (fps)	RMS of jitter (μs)	Minimum frame rate (fps)
K80, one GPU	9	282	47	260
K80, two GPUs	10	456	62	390
K80 + 2 × K20	4	541	54	460

5.2 Scaling when using several GPUs

When using two GPUs instead of one, the frame rate achieved increases by a factor of 1.6; when using four GPUs, it increases by about a factor 2.3. These factors are similar for different GPUs tested. The respective ideal increments would be by a factor of 2.0 and 4.0.

The main reason for the gain being lower than the ideal one is the kernel launching time; the overhead of the CPU handling an increased number of threads also has some contribution. If the GPUs are attached to the same PCIe bus, the data cannot be copied to the GPUs concurrently which further limits the gain in performance.

We investigated the kernel launching time and some options to reduce it. We introduced a mutex which enforces that, while all threads are performing step 1 in parallel (Table 1), only one thread at a time is performing steps 2 and 3. For some GPUs, this mutex improved the results, while for others it made them worse.

We investigated “dynamic parallelism,” a feature enabling a kernel running on a GPU to launch new kernels. Generally, the main advantage of this feature is the reduced communication between the GPU and the CPU in algorithms where the result of one step of the calculation leads to a decision about the next step of the calculation. Since there is no such communication in our algorithm, we only investigated whether it is advantageous to launch the kernels from the GPU rather than from the CPU. We launched one kernel from the CPU and that kernel then launched the three data processing kernels on the GPU. The result did not improve: the total kernel launching time remained the same.

6 Conclusions

We have implemented the real-time image processing for astronomical adaptive optics on GPUs. For an SCAO system of 80×80 subapertures, the maximum average frame rate achieved is about 1100 fps, using both GPUs of the NVidia K80 device. Using only one of the two GPUs, the frame rate achieved is about 700 fps. When running on more than one GPU, the gain in frame rate is limited by the kernel launching time.

The overhead of copying the camera images from CPU to GPU is largely reduced by processing the data in parallel to copying. The benefit of bypassing the CPU memory and copying the camera data to the GPU directly would be about 10–15% if using one GPU, and 20–25% when using two or four GPUs.

The distribution of cycle times (i.e., jitter) has a root-mean-square value of 20–30 μs and about one outlier in a million, with a value of up to 5 ms. The main source of jitter is the processing of data on the GPU; the contributions from data copying, kernel launching and CPU thread management are smaller.

With the addition of the cross-correlation algorithm, the frame rate achieved still exceeds 500 fps when using four GPUs.

These results demonstrate that GPUs are a good candidate for the RTC hardware for ELT instruments, although an additional step is needed to copy the camera images from the CPU memory to the GPU memory.

Acknowledgements We acknowledge the support from Science and Technology Facilities Council in the UK (ST/L00075X/1) and from the European Commission under program H2020-EU.1.2.2, Project 671662 (Green Flash), coordinated in H2020-FETHPC-2014.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Babcock, H.W.: The possibility of compensating astronomical seeing. PASP **65**(386), 229 (1953)
2. Davies, R., Kasper, M.: Adaptive optics for astronomy. Annu. Rev. Astron. Astrophys. **50**, 305–351 (2012)
3. de Zeeuw, T., Tamai, R., Liske, J.: Constructing the E-ELT. Messenger **158**, 3 (2014)
4. Dekany, R., et al.: PALM-3000: exoplanet adaptive optics for the 5 m Hale telescope. Astrophys. J. **776**, 130 (2013)
5. Vran, J.-P., et al.: Results of the NFIRAOS RTC trade study. Proc. of SPIE **9148**, 91482F (2014)
6. Barr, D., et al.: Reducing adaptive optics latency using Xeon Phi many-core processors. MNRAS **453**, 3222–3233 (2015)
7. Basden, A.G., Myers, R.: The Durham adaptive optics real-time controller: capability and extremely large telescope suitability. MNRAS **424**, 1483–1494 (2012)
8. Sevin, A., et al.: Enabling technologies for GPU driven adaptive optics real-time control. Proc. SPIE **9148**, 91482G (2014)
9. Thomas, S.J., et al.: Study of optimal wavefront sensing with elongated laser guide stars. MNRAS **387**(1), 173–187 (2008)
10. Poyneer, L.A., et al.: On-sky performance during verification and commissioning of the Gemini Planet Imager’s adaptive optics system. Proc. SPIE **9148**, 91480K-1 (2014)
11. Bitenc, U., et al.: On-sky tests of the CuReD and HWR fast wavefront reconstruction algorithms with CANARY. MNRAS **448**(2), 1199–1205 (2015)
12. Gendron, E., et al.: MOAO first on-sky demonstration with CANARY. Astron. Astrophys. **529**, L2 (2011)
13. Smith, M., et al.: Benchmarking hardware architecture candidates for the NFIRAOS real time controller. Proc. SPIE **9148**, 9148-2F-20 (2014)

Urban Bitenc is a postdoctoral research associate at the Centre for Advanced Instrumentation (CfAI) at the Physics Department of Durham University (UK), working in adaptive optics for astronomy. He obtained his degree in physics from the University of Ljubljana, Slovenia, in 2002. His Ph.D. (Ljubljana, 2007) was in experimental particle physics at the BELLE experiment, KEK institute, Japan. Then, he joined Freiburg University (Germany) to contribute to the background studies for the Higgs boson search at the ATLAS experiment at CERN. Afterward he worked as a systems engineer in a company, developing inertial navigation systems used in oil drilling, before he joined Durham University (UK) as a software engineer in the field of adaptive optics.

Alastair G. Basden has extensive expertise in low noise detectors and adaptive optics, including real-time control and simulation. He is an eternal postdoc at CfAI, Durham University (UK), where he has worked on AO systems for more than a decade.

Nigel A. Dipper is a senior research fellow at the Physics Department and the head of software at the CfAI, Durham University (UK). He obtained his Ph.D. at Southampton University (1980) and joined Durham University in 1985 to work on high-energy astrophysics. He specialized in software for astronomy in 1998, becoming the head of software for CfAI in 2003.

Richard M. Myers is a professor at the Physics Department, Durham University (UK), and the head of the Advanced Instrumentation Research Section. He obtained his Ph.D. at Durham University in 1988, became a lecturer in 1998, senior fellow in 2004, reader in 2008 and professor in 2012. His area of expertise is adaptive optics for astronomy.