Monthly Notices
of the
ROYAL ASTRONOMICAL SOCIETY

Mon. Not. R. Astron. Soc. **424,** 1483–1494 (2012) doi:10.1111/j.1365-2966.2012.21342.x

The Durham adaptive optics real-time controller: capability and Extremely Large Telescope suitability

A. G. Basden★ and R. M. Myers

Department of Physics, Durham University, South Road, Durham DH1 3LE

Accepted 2012 May 18. Received 2012 May 18; in original form 2012 April 26

ABSTRACT

The Durham adaptive optics real-time controller is a generic, high-performance real-time control system for astronomical adaptive optics systems. It has recently had new features added as well as performance improvements, and here we give details of these, as well as ways in which optimizations can be made for specific adaptive optics systems and hardware implementations. We also present new measurements that show how this real-time control system could be used with any existing adaptive optics system, and also show that when used with modern hardware, it has high enough performance to be used with most Extremely Large Telescope adaptive optics systems.

Key words: instrumentation: adaptive optics – instrumentation: high angular resolution – techniques: image processing.

1 INTRODUCTION

Adaptive optics (AO; Babcock 1953) is a technique for mitigating the degrading effects of atmospheric turbulence on the image quality of ground-based optical and near-infrared (near-IR) telescopes. It is critical to the high angular resolution performance of the next generation of Extremely Large Telescope (ELT) facilities, which will have primary mirror diameters of up to 40 m. Without mitigation, the general spatial resolution of such a telescope would be subject to the same atmospheric limitations as a 0.5-m diameter telescope. The proposed ELTs represent a large strategic investment and their successful operation depends on having a range of high-performance heterogeneous AO systems. As such, these telescopes will be the premium ground-based optical and near-IR astronomical facilities for the next two decades. The ELTs will, however, require a very significant extrapolation of the AO technologies currently deployed or under development for existing 4–10 m telescopes. Not least amongst the required developments of current AO technology is the area of real-time control. In this paper we describe the development and testing of a real-time controller, with the required scalability.

The Durham adaptive optics real-time controller (DARC; Basden et al. 2010a) is a real-time control system (RTCS) for AO that was initially developed to be used with the CANARY on-sky multi-object AO (MOAO) technology demonstrator (Gendron et al. 2011). As such, it was a significant success, being stable, configurable and powerful, and able to meet all the needs for this AO system. There was also demand for DARC to be used with other instruments, and so a further improved version of DARC was released to the public using an open source GNU General Public License (http://www.gnu.org/copyleft/gpl.html). Here, we provide information about the DARC platform, including new features, architectural changes, modularization, performance estimates, algorithm implementation and generalizations. We have tested the performance of this system in configurations matching a wide range of proposed ELT AO systems, and also configured to match proposed high-order 8-m class telescope AO systems, and a selection of results is presented here.

To date, each AO system commissioned on a telescope or used in a laboratory has generally had its own RTCS, leading to much duplicated effort. So far as we are aware the only other *multi-use* high-performance RTCS for AO is the European Southern Observatory (ESO) Standard Platform for Advanced Real-Time Applications (SPARTA; Fedrigo et al. 2006). Like DARC, SPARTA is designed to support heterogeneous components in high-performance configurations, including computational hardware other than standard PCs, and is currently being integrated with second generation Very Large Telescope (VLT) instruments such as Spectro-Polarimetric High-contrast Exoplanet REsearch (SPHERE; Fusco et al. 2006), GRound layer Adaptive optics Assisted by Lasers (GRAAL; Paufique et al. 2010) and Ground Atmospheric Layer Adaptive Corrector for Spectroscopic Imaging (GALACSI; Stuik et al. 2006), though it is not used outside ESO. The full SPARTA system cannot be used with just standard PCs and so expert programming skills are required, as well as dedicated software maintenance, and so would be unsuitable and costly for simple laboratory set-ups. A solution to this is to develop a system that is flexible enough to satisfy most performance requirements, is able to meet challenging AO system specifications using standard PC hardware, is simple to set up and use, and also supports hardware acceleration so that it is powerful enough to be used with demanding applications on-sky. DARC has been designed with these requirements in mind, and here we seek to demonstrate how it can be suited for most AO systems. A

★E-mail: a.g.basden@durham.ac.uk

common AO RTCS of this kind would be beneficial for the AO community, leading to a reduction in learning time and increased system familiarization.

This central processing unit (CPU) based approach to AO real-time control has not been successful in the past because it has been deemed that previously available CPUs have not been sufficiently powerful to meet the demanding performance requirements of on-sky AO systems (Fedrigo et al. 2006). The advent of multicore processors, which started to become commonly available from the mid-2000s, has been a key enabling factor for allowing our approach to succeed (our development commenced in 2008). It is now possible to obtain standard PC hardware with enough CPU cores to not only perform the essential real-time pipeline calculations [from wavefront sensor (WFS) data to deformable mirror (DM) commands], but also to perform necessary subtasks, such as parameter control, configuration and sharing of real-time system information and diagnostic streams. Attempts at using a single-core CPU for all these tasks have generally failed because context switching between these tasks has led to unacceptable jitter in the AO system. However, multicore systems do not suffer so much from this unpredictability. Previous systems have also not been freely available and have been closed source, which has greatly hindered uptake particularly for laboratory bench-based systems. Our approach does not have these restrictions.

Commercially available offerings, although impressive in many respects, do not scale well for use with future high-order AO system designs, and are often restricted to specific hardware, are designed for laboratory systems and can lack features required for high-order on-sky AO systems, such as pipelined pixel-stream processing.

There are three main areas of application for DARC: as a laboratory AO RTCS where flexibility and modularity are key, as well as stability; as a control system for instruments on 8–10 m class telescopes, where it is being evaluated for use with a number of AO systems currently under development; finally, as a control system for ELT instruments, where it is currently a potential candidate for use with two proposed AO systems.

In Section 2, we give an overview of the new features, including advanced algorithms, modularization, diagnostic data handling and tools available with DARC. In Section 3, we discuss how DARC can be optimized for use with a given AO system, and present some results demonstrating this optimization. In Section 4, we provide some examples of how DARC can be used with some existing and proposed demanding AO systems, and demonstrate the hardware that would be required for such operation. Finally, in Section 5 we draw our conclusions.

2 DARC FEATURES AND ALGORITHMS

There are several important architectural changes that have been made between the original version of DARC (Basden et al. 2010a) as used on-sky with CANARY and the current freely available version (to be used with future phases of CANARY) and these are discussed here. The changes include improved and expanded modularization, changes to diagnostic data handling, improvements to graphical processing unit (GPU) acceleration, the ability to be used asynchronously with multirate cameras, a generalization of pixel handling, advanced spot tracking algorithms, improved command line tools and the ability to use DARC in a massively parallelized fashion across multiple nodes in a computing grid. Hardware acceleration support has been improved principally through the increased modularization of DARC, and overall performance has also been dramatically improved.

DARC has also acquired the ability to allow user parameter changes on a frame-by-frame basis, allowing more complete control and dynamic optimization of the AO system. The control interface has added functionality that includes parameter subscription and notification, and greater control of diagnostic data, including redirection, partial subscription and averaging.

2.1 DARC modularization

Since conception, DARC has always had some degree of modularization; it has been possible to change cameras, DMs and reconstruction algorithms by dynamically loading and unloading modules into the real-time pipeline, without restarting DARC. This modularization has been extended to increase the degree of user customization that is possible with DARC. Module interfaces that allow modules to be dynamically loaded and unloaded in DARC now additionally include image calibration and wavefront slope computation interfaces, and an asynchronous open-loop DM figure sensing interface, as well as the pre-existing wavefront reconstruction interface. A parameter buffer interface has also been added, allowing customization of high-speed parameter input, facilitating the adjustment of any parameter on a frame-by-frame basis, thus allowing advanced use of the system, for example DM modulation and fast reference slope updating.

The user application programming interface (API) for developing DARC modules has been rationalized and now most modules include similar functions, which are called at well-defined points during the data-processing pipeline. This allows the developer to easily identify which functions are necessary for them to implement to achieve optimum AO loop performance, and encourages the consideration of algorithms that reduce latency and improve real-time performance.

Although it is possible to implement a large number of functions in each of these modules, typically they are not all required and so the developer should implement only the necessary subset for their particular application.

2.1.1 Module hook points

DARC uses a horizontal processing strategy as described by Basden et al. (2010a), which splits computational load as evenly as possible between available threads, thus allowing good CPU load balancing and high CPU resource utilization, giving low latency performance. To achieve this, each thread must be responsible for performing multiple algorithms, including WFS calibration, slope computation and partial wavefront reconstruction (rather than separate threads performing calibration, slope computation and reconstruction). To reconcile this with the modular nature of DARC, there are defined points at which module functions are called as shown in Fig. 1. A module developer then fills in the body of the module functions that they require. DARC will then call these functions at the appropriate time in a thread-safe way. Fig. 1 shows the DARC threading structure and the points at which module functions are called. It should be noted that the 'Process' function is called multiple times for each WFS frame until all the data have been processed (i.e. called for each subaperture in a Shack–Hartmann system). This approach has been taken to encourage a module developer to consider how their algorithm best fits into a low latency architecture, and to provide a consistent interface between modules. Unimplemented functions (those that are not required for a given algorithm) are simply ignored.

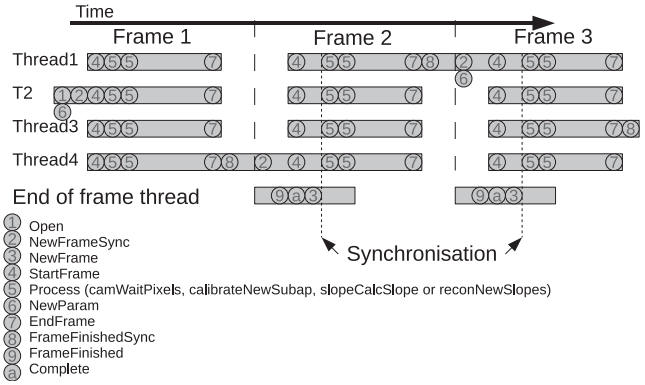

Figure 1. A demonstration of the points (in time) at which DARC module functions, if implemented, are called. Three WFS frames are shown with time increasing from left to right, with a new module being loaded at the start (point 1). This assumes that there are four main processing threads (Thread1, T2, Thread3 and Thread4), and the thread responsible for pre-processing and post-processing is also shown ('end of frame thread'). The legend shows the function names corresponding to the points (in time) at which the functions are called, for example, (1) is the point at which the module 'Open' function is called, thus initializing the module. The process function (5) is called repeatedly each frame until the frame has been processed (e.g. once for each subaperture). Further explanation is given in the main text.

We make a distinction between processing threads (labelled Thread1, Thread3, etc., in Fig. 1) which do the majority of the parallelized workload, and the 'end-of-frame' thread which is used to perform sequential workloads such as sending commands to a DM. Upon initialization of a module, the module 'Open' function is called by a single processing thread. Here, any initialization required is performed, such as allocating necessary memory, and (for a camera module) initializing cameras. Parameters (such as a control matrix or camera exposure time) are passed to the module using the DARC parameter buffer. To access this buffer, the 'NewParam' function is then called (if implemented). After this, the module is ready to use. One processing thread calls a 'NewFrameSync' function, for per-frame initialization. Each processing thread then calls a 'StartFrame' function, for per-thread, per-frame initialization. The 'Process' function is then called multiple times, while there are still data to be processed (for a Shack–Hartmann system this is typically once per subaperture, shared between the available processing threads). Once all such processing has been finished, each processing thread calls an 'EndFrame' function, which would typically perform gather operations to collate the results (e.g. summing together partial DM vectors). A single thread is then chosen to call a 'FrameFinishedSync' function to finalize this frame. After this function has been called, the 'end-of-frame' thread springs into life, allowing the processing threads to begin processing of the next frame, starting with the 'NewFrameSync' function. The 'end-of-frame' thread calls a 'FrameFinished' function for finalization during which (for a mirror module) commands should be send to the DM. A 'Complete' function is then called for each module, which can be used for initialization ready for the next frame. The 'end-of-frame' thread then calls a 'NewFrame' function. The 'end-of-frame' thread is not synchronized with the processing threads, and so there is no guarantee when the 'NewFrame' function is called relative to the functions called by the processing threads, except that processing threads will block before calling the 'Process' function until the 'end-of-frame' thread has finished.

Whenever the DARC parameter buffer is updated, the 'NewParam' function will be called by a single processing thread just before the 'NewFrameSync' function is called. When the module is no longer required (e.g. when the user wishes to try a new algorithm available in another module), a 'Close' function is called, to free resources.

The large number of functions may seem confusing, particularly since some appear to have similar functionality. Fortunately, most modules need only implement a small subset of these functions. The full suite of functions has been made available to give a module developer the required flexibility to create a module that is as efficient as possible, minimizing AO system latency.

Matrix operations are highly suited to this sort of horizontal processing strategy since they can usually be highly parallelized, and thus divided between the horizontal processing threads. Wavefront reconstruction using a standard matrix–vector multiplication algorithm (with a control matrix) is therefore ideally suited. Iterative wavefront reconstruction algorithms, for example those based around the conjugate gradient algorithms are less easy to parallelize, since each iteration step depends on the previous step. However, a horizontal processing strategy does allow the first iteration to be highly parallelized, which can lead to significant performance improvements when the number of iterations is small, for example when using appropriate pre-conditioners such as those used in the fractal iterative method (Béchet, Tallon & Thiébaut 2006). The post-processing (end-of-frame) thread (or threads) can then be used to compute the remaining iterations. Similarly, any system that requires multiple step reconstruction, for example a projection between two vector spaces, such as for true modal control, can be easily integrated.

2.1.2 DARC camera modules

An example of several simple camera modules is provided with the DARC source code. These are modules for which camera data are only available on a frame-by-frame basis (rather than a pixel-by-pixel basis), and typically would be used in a laboratory rather than on-sky. In this case, the camera data are transferred to DARC at the start of each frame (using the 'NewFrameSync' function), and other functions are not implemented (except for 'Open' and 'Close'). For such cameras, there is no interleaving of camera read-out and pixel processing, and so a higher latency results.

For camera drivers that have the ability to provide pixel-stream access (i.e. the ability to transfer part of a frame to the computer before the detector read-out has finished), a more advanced camera module can be implemented, and examples are provided with DARC. Such modules allow interleaving of camera read-out with processing, making use of the 'Process' function to block until enough pixels have arrived for a given WFS subaperture, after which calibration and computation of this subaperture can proceed.

2.1.3 Parameter updating

DARC has the ability to update any parameter on a frame-by-frame basis. However, since this could mean a large computational requirement (to compute the parameters from available data), or a large data bandwidth requirement (e.g. for updating a control matrix), this update ability is implemented using a DARC module interface. The users can then create such a module depending on their specifications to best suit the needs and requirements of their systems, for example using a proprietary interconnect to send the parameters. A standard set of DARC functions are provided, which should be overwritten for this parameter buffer interface to take

effect. This buffer module is again dynamically loadable, meaning that this ability can easily be switched on and off.

This ability to update parameters every frame or in a deterministic fashion (e.g. at a given frame number) is optional, and additional to the non-real-time parameter update facility that is used to control DARC as discussed by Basden et al. (2010a), which allows parameters to be changed in a non-deterministic fashion (non-real-time, i.e. there is no guarantee that a parameter will be changed at a particular frame number or time).

2.2 Diagnostic data handling

The general concept of diagnostic data handling is described in the original DARC paper (Basden et al. 2010a). In summary, there is a separate diagnostic stream for each diagnostic data type (raw pixel data, calibrated pixel data, wavefront slope measurements, mirror demands, etc.). It should be noted that diagnostic data handling is used only to provide data streams to clients, not for the transfer of data along the real-time pipeline. As such, the diagnostic data system does not need to be hard-real-time.

Transport of DARC diagnostic data uses Transmission Control Protocol/Internet Protocol (TCP/IP) sockets by default, which lead to a reliable, simple and fairly high-performance system that is easy to understand and set up, with minimal software configuration and installation. However, because DARC is modular by design, users are able to replace this system with their own, should they have a need to, using their own transport system and protocols to distribute these data to clients. This is well suited to a facility class telescope environment, where standardized protocols must be followed.

2.2.1 Default diagnostic data implementation

The default DARC diagnostic system seeks to minimize network bandwidth as much as possible using point-to-point (PTP) connections. Diagnostic data are sent from the real-time system to a remote computer, where the data are written into a shared memory circular buffer. Clients on this computer can then access the data by reading from the circular buffer, rather than requesting data directly from the real-time system across the network. Additionally, these data can then be redistributed to further remote computers, allowing the data to be read by other clients here, as shown in Fig. 2. Hence, each diagnostic stream needs to be sent only once (or, depending on network topology, a small number of times) from the main real-time

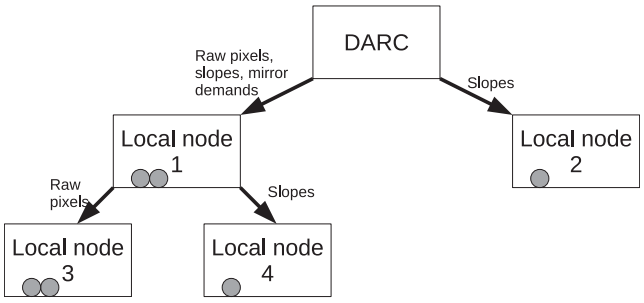

Figure 2. A demonstration of the DARC diagnostic data system. Here, raw pixel, slope and mirror data are being sent from the real-time part of the system to one local node, which is in turn sending raw pixels and slopes to other nodes. A further local node is receiving slopes directly from the real-time part of the system. Clients that process these data are represented by grey circles.

computer (rather than once per client), and network bandwidth can be tightly controlled.

In cases where only part of a diagnostic stream is required (e.g. pixel data from a single camera in a multicamera system, or a subregion of an image), these data can be extracted by a DARC client into a new diagnostic stream before being distributed over the network, reducing bandwidth requirements. Additionally, for cases where only the sum or average of many frames of data from a given stream is required, this operation can be performed using a client provided by DARC before data are transported over the network, again greatly saving network bandwidth, for example for image calibration.

It should be noted that the default diagnostic system does not use broadcasting or multicasting. This is because broadcasting and (in its simplest form) multicasting are inherently unreliable and thus would not provide a reliable diagnostic stream, giving no guarantee that data would reach their destination, which is undesirable for an AO system. Although reliable multicasting protocols are available, a decision has been made not to use these by default, because this would increase the complexity of DARC, and would be unnecessary for users of simple systems. However, we would like to reiterate that such diagnostic systems can easily be added and integrated with DARC by the end user.

On the real-time computer and remote nodes with diagnostic clients, a region of shared memory (in /dev/shm) is used for each diagnostic stream, implemented using a self-describing circular buffer typically hundreds of entries long (though this is configurable and in practice will depend on available memory and stream size). Streams can be individually turned on and off as required using the DARC control interface (using a graphical, script or command line client or the API). DARC will write diagnostic data to the circular buffers of streams that are not switched off, and these data can then be read by clients or transported. The rate at which DARC writes these data can be changed (every frame, or every n frames with a different rate for each diagnostic stream). Clients can then retrieve as much or as little of these data as they require, by setting the subsampling level at which they wish to receive.

Since TCP/IP is used, retransmission of data may be necessary when network hiccups occur (though this is handled by the operating system). The processes responsible for sending data over the network to clients will block until an acknowledgment from the client has been received (this is handled by the operating system), and therefore at times may block for a longer than average period while waiting for retransmission. If this happens frequently enough (e.g. on a congested network), then the head of the circular buffer (the location at which new diagnostic data are written to the circular buffer) will catch up with the tail of the buffer (the location at which data are sent from). To avoid data corruption, the DARC sending process will jump back to the head of the circular buffer once the tail of the buffer falls more than 75 per cent of the buffer behind the head. Therefore, a chunk of frames will be lost. This is undesirable, but should be compared with what would occur with an unreliable protocol, for example User Datagram Protocol (UDP). Here, the sending process would not be blocked, and so would keep up with the head of the circular buffer. However, when a packet fails to reach its destination, due to congestion or a network hiccup, the packet would simply be dropped and not retransmitted. Therefore, there would be a portion of a frame of data missing (corresponding to the dropped packet), rendering (in most cases) the entire frame unusable. We should therefore consider two cases: in a highly loaded network, it is likely that the number of partial (unusable) frames received would be greater than the number of frames dropped when

using a reliable protocol even though the total network throughput would be greater, because dropped packets would be dispersed more-or-less randomly affecting a greater number of frames, while dropped frames would be in chunks. In a less loaded network, UDP packets would still be occasionally dropped, resulting in unusable frames, while a reliable protocol would (on average) be able to keep up with the circular buffer head, dropping behind occasionally, but not far enough to warrant dropping a chunk of frames to jump back to the circular buffer head.

2.2.2 Disadvantages

The disadvantage of this simple approach to diagnostic data is that TCP/IP is unicast and PTP, meaning that if multiple clients on different computers are interested in the same data then the data are sent multiple times, a large overhead. However, we take the view that the simplicity of the default system outweighs the disadvantages, and that for more advanced systems, a separate telemetry system should be implemented for which we are unable to anticipate the requirements.

2.2.3 Diagnostic feedback

We have so far discussed the ability to propagate real-time data to interested clients. However, it is often the case that these data will be processed and then injected back into the real-time system on a per-frame basis, useful not only for testing, but also for calibration tools such as turbulence profiling or DM shape feedback. This feedback interface module can be implemented in DARC in several ways. One option is to use the per-frame parameter update module interface as discussed previously, provided by the user to suit their requirements. Another option is to modify an existing DARC processing module (dynamically loadable), to accept the expected input, for example modify a wavefront reconstruction module to accept an additional input (via InfiniBand or any desired communication protocol) of actual DM shape, to be used for pseudo-open-loop control reconstruction.

2.3 User facilities

The DARC package comes with an extensive suite of user tools, designed to simplify the set-up and configuration of DARC. These include command-line-based tools, a graphical interface, a configurable live display tool and an API. Since the DARC control interface is based upon Common Object Request Broker Architecture (CORBA; http://www.omg.org/cgi-bin/doc?formal/0010-33.pdf), a client can be written in any programming language that has a suitable CORBA implementation.

Using these tools allows additional customized packages and facilities to be built, specific for the AO system in question. An example of this would be a tip–tilt offload system, which would capture slope diagnostic data from DARC, and if mean slope measurements became too large would inform the telescope to update its tracking. Many such systems based around DARC diagnostic data have been used successfully with CANARY.

2.4 Configurable displays

DARC does not know the nature of the data in the diagnostic streams that it produces. It is known, for example, that a particular stream contains DM demand data, and how many DM actuators there are; however, DARC knows nothing about the mapping of these actuators on to physical DMs, how many DMs there are, and what geometry they have. Therefore, to display these data in a way meaningful to the user, additional information is required.

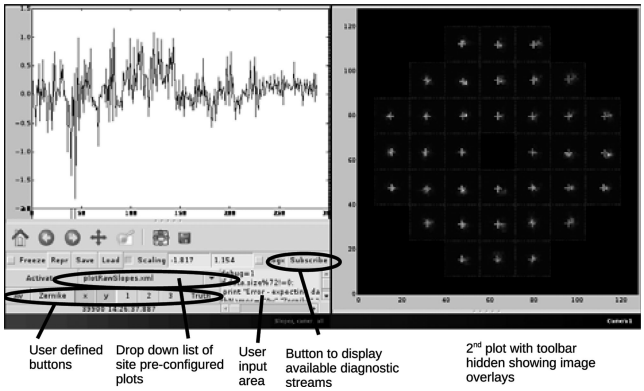

Figure 3. Two instances of the DARC live display tool. On the left-hand side a one-dimensional display of slope measurements is shown, along with the associated tool bar, configured for this AO system (CANARY) allowing the user to select which slopes from which WFS to display. On the right-hand side, a two-dimensional WFS image is displayed, with subaperture boundaries and current spot locations overlain (as a grid pattern and cross hairs, respectively), and the display is receiving multiple diagnostic streams (image and slopes) simultaneously. In this display, the tool bar is hidden (revealed with a mouse click).

The DARC live display tool allows user configuration, both manually entered and via configuration files. Additionally, a collection of configuration files can be used and the live display will present a selection dialogue box for the user to rapidly switch between configurations, for example to display a phase map of different DMs, or to switch between a phase map and a WFS image display. Each configuration can specify which diagnostic streams should be received and at what rate, allowing a given configuration to receive multiple diagnostic streams simultaneously, for example, allowing a spot centroid position to be overlain on a calibrated pixel display. Manually entered configuration is useful while an AO system is being designed and built, and can be used for example to change a one-dimensional pixel stream into a two-dimensional image for display.

The live display can be configured with user-selectable buttons, which can then be used to control this display configuration. For example, when configured for a WFS pixel display, the user might be able to turn on and off a vector field of the slope measurements by toggling a button.

This configurability is aimed at ease of use, as a simple way of getting a user-friendly AO system up and running. Although not designed to be used as a facility class display tool, it does have sufficient flexibility and capability to function as such. Fig. 3 shows two instances of the display tool being used to display WFS slope measurements and calibrated pixel data.

2.5 DARC algorithms

Since DARC is primarily based around CPU computation and is modular, it is easy to implement and test new algorithms. As a result, there are a large number of algorithms implemented, and this list is growing as new ideas or requirements arise. Some of these algorithms are given by Basden et al. (2010a), and we present additional algorithms here.

2.5.1 Pixel processing and slope computation

The use of the brightest pixel selection algorithm (Basden, Myers & Gendron 2011) has been demonstrated successfully on-sky using DARC. This algorithm involves selecting a user-determined number of brightest pixels in each subaperture and setting the image threshold for this subaperture at this level. This helps to reduce the impact of detector read-out noise, and can lead to a significant reduction in wavefront error.

The ability of DARC to perform adaptive windowing, or spot tracking with Shack–Hartmann WFSs, has been mentioned previously (Basden et al. 2010a). Further functionality has now been added allowing groups of subapertures to be specified, for which the adaptive window positions are computed based on the mean slope measurements for this group, and hence these grouped subapertures all move together. This allows, for example, tip–tilt tracking on a per-camera basis when multiple WFSs are combined on to the same detector.

One danger with adaptive windowing is that in the event of a spurious signal (e.g. a cosmic ray event) or if the signal gets lost (e.g. intermittent cloud), then the adaptive windows can move away from the true spot location. Adaptive window locations will then be updated based upon noise, and so the windows will move randomly until they find their Shack–Hartmann spot, or fix on another nearby spot. To prevent this from happening, it is possible to specify the maximum amount by which each window is allowed to move from the nominal spot location. Additionally, adaptive window locations are computed from the local spot position using an infinite impulse response (IIR) filter, for which the gain can be specified by the user, helping to reduce the impact of spurious signals.

In addition to weighted centre of gravity and correlation-based slope computation, a matched filter algorithm can also be used. DARC can be used not only with Shack–Hartmann WFSs, but also with Pyramid WFSs and, in theory, with curvature WFSs (though this has never been tested), due to the flexible method by which pixels are assigned to subapertures (which can be done in an arbitrary fashion). The modular nature of DARC means that other sensor types could easily be added, for example Yet Another Wavefront sensor (YAW; Gendron et al. 2010) and optically binned Shack–Hartmann sensors (Basden et al. 2007).

2.5.2 Reconstruction and mirror control

In addition to matrix–vector based wavefront reconstruction (allowing for least-squares and minimum mean square error algorithms), linear quadratic Gaussian (LQG) reconstruction can also be carried out, allowing for vibration suppression, which can lead to a significant performance improvement (Correia et al. 2010). An iterative solver based on preconditioned conjugate gradient is also available, and can be used with both sparse and dense systems. Using this reconstruction technique (Gilles, Ellerbroek & Vogel 2003) has the advantage that a matrix inversion to compute a control matrix is not required. In addition, an open-loop control formulation is available which allows DM commands (a) to be computed according to

$$a_i = (1 - g)\mathbf{E} \cdot a_{i-1} + g\mathbf{R} \cdot s_i, \quad (1)$$

where s_i are the current wavefront slope measurements, g is a gain parameter, $\mathbf{R}$ is the control matrix and $\mathbf{E}$ is a square matrix, which for an integrator control law would be equal to an identity matrix scaled by $1/(1 - g)$.

DARC also contains the ability to perform automatic loop opening in the case of actuator saturation, i.e. to automatically open the control loop and flatten the DM if a pre-defined number of actuators reach a pre-defined saturation value. This can be important to avoid damage while testing new algorithms and control laws.

Some DMs display hysteresis and other non-linear behaviour that results in the shaped DM not forming quite the shape that was requested. To help reduce this effect, DARC includes the option to perform actuator oscillation around the desired DM shape position, allowing the effect of hysteresis to be greatly reduced. Typically, a decaying sine wave is used, with the decay leading to the desired position.

Since DARC has the ability to update parameters on a frame-by-frame basis, it has the ability to modulate (or apply any time-varying signal to) some or all of the DM actuator demands, allowing complex system operations to be performed.

2.5.3 Advanced computation

DARC has the ability to operate multiple WFSs asynchronously, i.e. at independent frame rates (which are not required to be multiples of each other). This gives the ability to optimize WFS frame rate depending on guide star brightness, and so can lead to an improvement of AO system performance. This capability is achieved by using multiple instances of DARC, one for each WFS, which compute partial DM commands based on the WFS data, and a further instance of DARC which combines the partial DM commands once they are ready, using shared memory and mutual exclusion locks for interprocess communication. The way in which the partial DM commands are combined is flexible, with the most common option being to combine these partial commands together as they become available and then update the DM, meaning that the DM is always updated with minimal latency.

The ability to operate multiple instances of DARC, combined with modularity, means that DARC can be used in a distributed fashion, allowing computational load to be spread over a computing grid. To achieve this, modules responsible for distributing or collating data at different points in the computation pipeline are used, with data being communicated over the most efficient transport mechanism. At present, such modules exist based on standard Internet sockets, and also using shared memory. This flexibility allows DARC to be optimized for available hardware. Extremely demanding system requirements can therefore be met.

2.6 Hardware acceleration

The modular nature of DARC makes it ideal for use with acceleration hardware, such as field programmable gate arrays (FPGAs) and GPUs. Two hardware acceleration modules currently exist for DARC (and of course more can easily be added). These are an FPGA-based pixel-processing unit that can perform image calibration and optionally wavefront slope calculation and is described elsewhere (Fedrigo et al. 2006), and a GPU-based wavefront reconstruction module, which we now describe.

2.6.1 GPU wavefront reconstruction

The GPU wavefront reconstruction module can be used with any Compute Unified Device Architecture (CUDA) compatible GPU. It performs a matrix–vector multiplication-based wavefront reconstruction, which (depending on the matrix) includes least squares and minimum variance reconstruction. As discussed previously,

DARC uses a horizontal processing strategy processing subapertures as the corresponding pixel data become available. The GPU reconstruction module also follows this strategy, with partial reconstructions (using appropriate subsets of the matrix) being performed before combination to provide the reconstructed wavefront.

The control matrix is stored in GPU memory, and wavefront slope measurements are uploaded to the GPU every frame. The GPU kernel that performs the operations has been written specifically for DARC, providing a 70 per cent performance improvement over the standard CUDA Basic Linear Algebra Subroutines (cuBLAS) library which is available from the GPU manufacturer NVIDIA. This module uses single precision floating point data.

The performance reached by this module is limited by GPU internal memory bandwidth rather than computation power, since the matrix has to be read from GPU memory into GPU processors every frame. We are able to reach about 70 per cent of peak theoretical performance on a NVIDIA Tesla 2070 GPU card. An alternative version of this DARC module also exists which stores the control matrix in a 16-bit integer format, with conversion to single precision floating point performed each frame before multiplication. This allows a performance improvement of about 80 per cent (reducing computation time by 45 per cent) with a trade-off of reduced precision. However, as shown by Basden, Myers & Butterley (2010b), 16-bit precision is certainly sufficient for some ELT scale AO systems (it may not be sufficient for a high contrast system, though we have not investigated this). It should be noted that slope measurements for astronomical AO systems are typically accurate to at most 10–11 bits of precision, limited by photon noise.

3 DARC OPTIMIZATION

DARC has been designed to provide low latency control for AO, operating with a baseline of minimal hardware (a computer), whilst also providing the ability to use high end hardware, and hardware acceleration. This has been achieved by careful management of the workload given to processor threads using a horizontal processing strategy, and by reducing the need for synchronization, as described by Basden et al. (2010a). DARC has since been updated to further reduce this latency, with steps being taken to reduce the number of thread synchronization points (thus reducing synchronization delays), and also providing control over where post-processing is performed. The increased modularization of DARC has led to a clearer code structure and allowed the thread synchronization points to be rationalized, providing opportunities for better optimization of modules, which can be fitted into the DARC structure more easily.

3.1 Site optimization

To optimize DARC for a specific application, there are several steps that can be taken to allow the best performance (lowest latency) to be achieved for a given hardware set-up. For all of these steps, no recompilation is necessary, and many can be performed without stopping DARC. In this section we will present these optimizations and discuss why these can make a difference, and how these should be applied. The large number of optimization points built into DARC mean that it is well suited to meet the demands of future instruments.

3.1.1 Number of threads

The main way to optimize DARC performance is to adjust the number of processing threads used. A higher order AO system will have reduced latency when more processing threads are used; however, the number of threads should be less than the number of processing cores available, which will in turn depend on the processing hardware. The balance between processing power and memory bandwidth can also affect the optimal number of processing threads. Fig. 4 shows how the maximum achievable frame rate is affected by the number of processing threads for a 40 × 40 subaperture single conjugate adaptive optics (SCAO) system, and it should be noted that a higher frame rate corresponds to lower latency. The frame rates displayed here were measured using a computer with two six-core Intel E5645 processors with a clock speed of 2.4 GHz and hyper-threading enabled, giving a total of 24 processing cores (12 physical). We have restricted threads to run on a single core (i.e. no thread migration, by setting the thread affinity), with the first six threads running on the first CPU, the next six the second CPU, the next six on the first and so on as required (so each processor core may have multiple threads, but each thread is only allowed on a single core). It can be seen from this figure that DARC gives near-linear performance scaling with number of threads up to the number equal to the number of physical CPU cores (12), at which point, maximum performance is achieved. This is followed by a dip at 13 and 14 threads as one core is then having to run two DARC threads. Performance then increases again up to 24 threads (except for a dip at 22 threads, which we are unsure about, but which is repeatable), after which performance levels off (and eventually falls) as each core has to run an increasing number of threads, and thread synchronization then begins to take its toll.

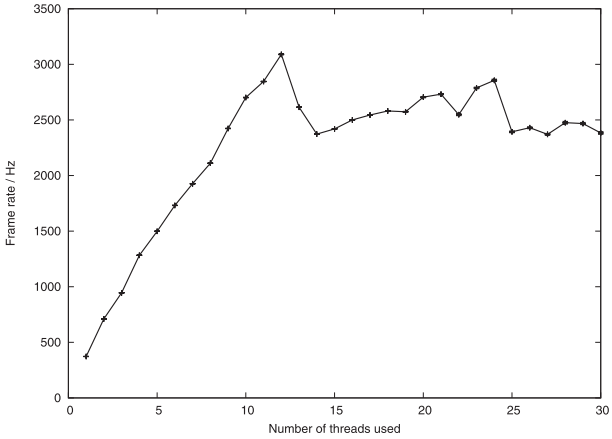

Figure 4. A plot showing how the AO system maximum achievable frame rate depends on the number of DARC processing threads used. Uncertainties are shown but generally too small to discern.

This figure shows that hyper-threading is detrimental to performance in this case, and we recommend that the use of hyper-threading should be investigated by any DARC implementer and switched off (in the computer BIOS) if it is detrimental. However, this is not a condition that we would wish to impose because all situations are different and some users may find it desirable to have hyper-threading.

It should be noted that there is also the option of using a separate thread for initialization and post-processing of data, or to have this work done by one of the main processing threads, and each option can provide better performance for different situations.

3.1.2 Affinity and priority of threads

By giving processing threads elevated priorities, the Linux kernel will put more effort into running these threads. However, when

using DARC, the best performance is not necessarily achieved by giving all threads high (or maximum) priority. Rather, there can be an advantage in considering the work that each individual thread is required to do, and whether particular threads are likely to be on the critical path at any given point.

As a simple example, consider the case of an AO system comprising a high-order WFS and a tip–tilt sensor. In DARC, one thread would be assigned to the tip–tilt sensor, which has very little work to do, while a number of threads would be assigned to the high-order WFS, each of which will have more computational demand. There will also be a post-processing thread which is used for operations that are not suitable for multithreaded use, for example sending mirror demands to a DM. In this case, lowest latency will be achieved by giving highest priority to the high-order WFS processing threads. The tip–tilt thread should be given a lower priority, and its work will be completed during computation gaps. The post-processing thread should be given a higher priority, so that it will complete as soon as all data are available, reducing latency.

The location of threads can also be specified, restricting a given thread to run on one, or a subset of CPU cores. This prevents the kernel from migrating threads to different cores, and also allows threads to be placed closest to hardware in non-uniform systems (e.g. where one CPU has direct access to an interface bus), so improving performance. A fine tuning of latency and jitter can be achieved in this way.

3.1.3 Subaperture numbers and ordering

Another optimization that can be made, but which is far less obvious, is to ensure that there are an even number of subapertures defined for each WFS, and that pairs of subapertures are processed by the same processing thread (ensured using the subaperture allocation facility). This ensures that wavefront slopes are aligned on a 16-byte memory boundary for the partial matrix–vector multiplication during the wavefront reconstruction processing, and allows streaming single instruction, multiple data (SIMD) extension (SSE) operations (vector operations) to be carried out. It should be noted that if the system contains an odd number of subapertures, an additional one can be added that has no impact on the final DM calculations simply by adding a column of zeros to a control matrix. This optimization can have a large impact on performance. Where possible, all matrix and vector operations in DARC are carried out using data aligned to a 16-byte boundary to make the maximum use of SSE operations.

3.1.4 Pixel read-out and handling

A low latency AO system will usually use a WFS camera for which pixels can be made available for processing as they are read out of the camera, rather than on a frame-by-frame basis, or at least made available in chunks smaller than a frame. This allows processing to begin before the full frame is available, and since camera read-out is generally slow, this can give a significant latency reduction, and can mean that minimal operations are required once the last pixel arrives. With DARC, optimizations can be made by optimizing the 'chunk' size, i.e. the number of pixels that are made available for processing together. A smaller chunk size will require a larger number of interrupts to be raised, and also a larger number of data requests [typically direct memory access (DMA)]. Conversely, a larger chunk size will mean that there is a greater delay between pixels leaving the camera and becoming available for processing. There is therefore a trade-off to be made, which will depend on camera type, data acquisition type and processor performance among other things.

Related to this is an optimization that allows DARC to process subapertures in groups. There is a parameter that is used by DARC to specify the number of pixels that must have arrived before a given subaperture can be processed. If this value is rounded up to the nearest multiple of chunk size then there will be multiple subapertures waiting for the same number of pixels to have arrived, and hence these can be processed together, reducing the number of function calls required, and hence the latency. The processing of particular subapertures can be assigned to particular threads to help facilitate this optimization.

3.1.5 Linux kernel impact

The Linux kernel version with which DARC is run can also have an impact on latency. We do not have a definitive answer to which is the best kernel to use, because this depends somewhat on the system hardware, and also AO system order. We also find that when using a real-time kernel (with the RT-preempt patch), latency (as well as jitter) is slightly reduced. Therefore, if DARC is struggling to reach the desired latency for a given system, investigating different kernels may prove fruitful. This can also impact diagnostic bandwidth too.

3.1.6 Grid utilization

More ambitious latency reduction can be achieved by spreading the DARC computational load across a grid computing cluster. For some AO systems, the division of work will fall naturally. For example, systems with more than one WFS could place each WFS on a separate grid node, before combining the results in a further node that sends commands to a DM. For other AO systems, the division of labour may not be so obvious, for example an extreme adaptive optics (XAO) system with only a single WFS. In cases such as this, separation would be on a per subaperture basis with responsibility for processing different subapertures being placed on different grid nodes. However, to achieve optimal performance in this case it must be possible to split WFS camera pixels between nodes using for example a cable splitter, rather than reading the WFS into one node and then distributing pixels from there, which would introduce additional latency.

The effectiveness of using DARC in a grid computing environment depends to some extent on the communication link between grid nodes. Real-time data must be passed between these nodes, and so we recommend that dedicated links be used, which will not be used for other communications, such as diagnostic data. These links should also be deterministic to reduce system jitter. Additionally, the higher the performance of these links, the lower the overall latency will be. A DARC implementer will need to implement their own DARC modules according to the communication protocol and hardware that they use, as the standard DARC package contains only a TCP/IP implementation that is not ideal for low jitter requirements. It will also be necessary for the implementer to ensure that a lower latency is achieved when using a grid of computers than can be achieved on a single computer.

4 DARC IMPLEMENTATIONS

Optimizing DARC for a particular AO system requires some thought. Using hardware available at Durham, consisting of a dual

Table 1. Details of some existing and proposed AO systems with demanding computational requirements for the real-time system. The high degree of correction makes these systems cutting-edge in the science they can deliver.

System	Wavefront sensor	Deformable mirror	Frame rate	Telescope size
VLT planet-finder	40 × 40	41 × 41	2 kHz	8 m
Palm-3000	62 × 62	3300	2 kHz	5 m
ELT-EAGLE	11 of 84 × 84	20 of 85 × 85	250 Hz	∼40 m

six-core processor server (Intel E5645 processors with a clock speed of 2.4 GHz) with three NVIDIA Tesla 2070 GPU acceleration cards, we have implemented the real-time control component of several cutting edge existing and proposed AO systems, as given in Table 1. In doing this, we have sought to minimize the latency that can be achieved using DARC for these systems. In the following sections we discuss these implementations, the achievable performance and the implications that this has.

4.1 DARC as a RTC for system resembling a VLT planet-finder

A planet-finder class instrument is currently under development for one of the VLTs in Chile. The AO system for this instrument, SPHERE (Fusco et al. 2006), is based on a 40 × 40 subaperture Shack–Hartmann extreme AO system. Real-time control will be provided by an ESO standard SPARTA system, comprised of a Xilinx Virtex II-Pro FPGA for pixel processing (WFS calibration and slope calculation), and a digital signal processor (DSP)-based system for wavefront reconstruction and mirror control, comprised of four modules of eight Bittware Tigershark TS201 DSPs.

This AO system is required to operate at a frame rate of at least 1.2 kHz, and a goal of 2 kHz, with a total latency (including detector read-out and mirror drive) of less than 1 ms (Fusco et al. 2006).

We have implemented an equivalent AO RTCS using DARC, based on the aforementioned server PC, but without using the GPU acceleration cards. It should be noted that we do not have an appropriate camera or DM to model this system, and so the implementation here does not include these system aspects; however, it does include all other aspects of a RTCS, including interleaved processing and read-out and thread synchronization. We are able to operate this system at a maximum frame rate of over 3 kHz, using 12 CPU threads, corresponding to a total mean frame processing time of about 323 μs, as shown in Fig. 4. In a real system (with a real camera), WFS read-out would be interleaved with processing, and so the latency, defined in this paper as the time taken from the last pixel received to the last DM actuator set, would be significantly less than this, because most processing would occur while waiting for pixels to be read out of a camera and sent to the RTCS. Therefore, the latency measured from the last pixel acquired to the last command out of the box is expected to be well below 100 μs, well within the required specification. Here, we find that processing subapertures in blocks of about 40 at a time gives the best performance. When interpreting these results, it should be noted that in a standard configuration as used here, DARC does not allow pipelining of frames: computation of one frame must complete before the next frame begins, and so the maximum frame rate achievable is the inverse of the frame computation time.

DARC performance for this configuration was assessed by using the Linux real-time clock to measure frame computation time.

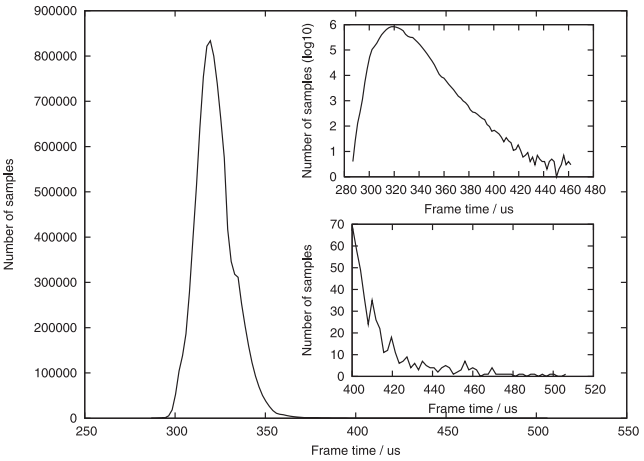

Figure 5. A histogram of frame computation times for a 40 × 40 subaperture AO system measured with 10^7 samples. Insets show a logarithmic histogram (showing outliers more clearly), and also a linear histogram of outliers only. This clearly shows that jitter is well constrained.

A real-time Linux kernel (2.6.31-10.rt, available from Ubuntu archives) was used. The frame computation time was measured to be 323 ± 11 μs, averaged over 10 million consecutive frames, a histogram of which is shown in Fig. 5. The system jitter of 11 μs rms was measured (the standard deviation of frame computation time). The maximum frame time measured over this period was 508 μs, though this (as can be seen from the standard deviation) was an extremely rare event. In fact, only 501 frames (out of 10^7) took longer than 400 μs to compute, only 106 frames took longer than 425 μs and only two frames took longer than 500 μs. The mean frame computation time here corresponds to a frame rate of greater than 3 kHz.

Equivalent measurements made with a stock Linux kernel (2.6.32) did not give significantly worse performance, with the mean frame computation time increasing slightly to 343 ± 13 μs and no increase in the processing time tail.

The introduction of a camera and a DM to this system would increase the frame computation time due to the necessity to transfer pixel data and DM demand data; however, our experience shows that we would not expect a significant increase in computation time, and maximum frame rates greater than 2 kHz would still be easily achievable.

It is interesting to note that more recent stock Ubuntu kernels (2.6.35 and 2.6.38) give far worse performance, almost doubling the frame computation time. At this point we have not investigated further, though intend to do so. This could be due to parameters used during kernel compilation, or due to actual changes in the kernel source, though other available bench marks do not suggest that this is a likely problem. However, it is worth bearing in mind that the performance of a software-based AO control system may be dependent on the operating system kernel used.

Typically, a design for an AO RTCS is made before actual hardware and software is available, and so predictability of performance of a CPU-based real-time controller is not usually well defined.

However, by using pre-existing real-time control software such as DARC, performance predictability can be improved, as it allows hardware to be purchased earlier in the development cycle of a RTCS, for immediate use. This removes much of the uncertainty of CPU-based controllers much earlier in the design and prototyping phases of AO system development.

4.2 DARC as a RTC for a system resembling the Palm-3000 AO system

The Palm-3000 AO system on the 5-m Hale telescope is the highest order AO system yet commissioned (Truong et al. 2008), and has the highest computational demands. At highest order correction, it uses a 62 × 62 Shack–Hartmann WFS and a DM with about 3300 active elements, and is specified to have a maximum frame rate of 2 kHz. For real-time control, this system uses eight PCs and 16 GPUs, far more hardware than we have available at our disposal. However, by using DARC on our existing (previously mentioned) hardware, we have been able to meet this specification, though again, as we do not have suitable cameras and DMs, our measurements do not include these.

In order to implement this system, we use the three Tesla GPUs for wavefront reconstruction, spreading the matrix–vector multiplication equally between them. The matrix in each GPU has dimensions equal to $N_{act} \times N_{slopes}/3$, with N_{act} being the number of DM actuators and N_{slopes} being the number of slopes (twice the number of active subapertures). In order to interleave camera read-out with pixel processing, we split these multiplications into four blocks, i.e. perform a partial matrix–vector multiplication once a quarter, a half, three quarters and all the slope measurements for each GPU have been computed, thus performing a total of 12 matrix–vector multiplications per frame. It is interesting to note that the actual Palm-3000 RTCS splits processing into two blocks per GPU, i.e. processing occurs halfway through pixel arrival and after all pixels have arrived. Our implementation performs pixel calibration and slope calculation in CPU, dividing the work equally between 12 processor cores using 12 processing threads.

By configuring DARC in this way, we are able to achieve a frame rate of 2 kHz and a corresponding frame processing time of 500 μs. The addition of a real camera and DM would increase this processing time meaning that the official specifications would not be met. However by adding a fourth GPU, performance could be increased.

It should be noted that our GPUs are of a higher specification, with memory bandwidth (the bottleneck) being 65 per cent higher than that used by Palm-3000, which will account for some of the difference. We also perform calibration and slope computation in CPU, while the Palm-3000 system performs this in GPU, and the Palm-3000 system will include some overhead for contingency (i.e. the maximum frame rate is likely to be greater than 2 kHz should the WFS allow it). Because all our calculations are performed within one computing node, we do not suffer from increased latency due to computer–computer communication, which the Palm-3000 system will include. These differences help to explain how DARC is able to perform the same task using significantly less hardware and a simpler design.

This demonstrates that DARC is suitable for use with high-order AO correction, despite being primarily CPU based, i.e. CPU-based RTCSs are sufficiently powerful for current AO systems.

4.3 DARC as a RTC for a system resembling E-ELT EAGLE

A multi-object spectrograph for the European ELT (E-ELT) is currently in the design phase. This instrument, EAGLE (Cuby et al. 2008), will have a multi-object AO system (Rousset et al. 2010), with independent wavefront correction along multiple lines of sight, in directions not necessarily aligned with WFSs, as shown in Fig. 6. A current design for EAGLE consists of 11 WFSs (of which six use laser guide stars), and up to 20 correction arms. Each WFS has

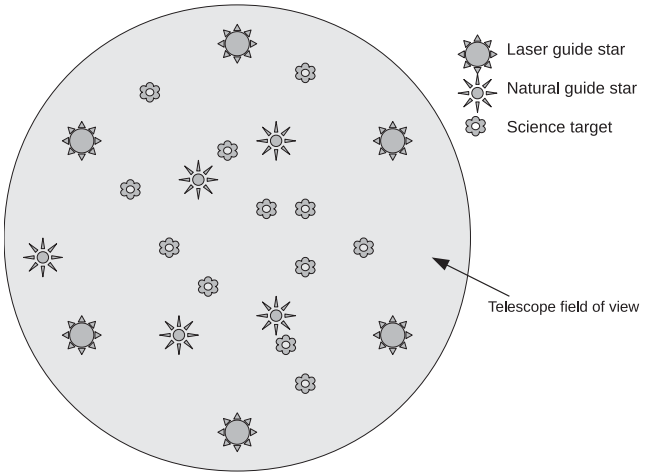

Figure 6. A demonstration of multi-object AO. The large circle represents the telescope field of view, and six laser guide stars and associated WFSs are arranged in a ring near the edge of this. Five natural guide star WFSs are then placed on appropriate stars, and multiple science targets, each with a DM, can then be picked out.

84 × 84 subapertures, and each correction arm contains an 85 × 85 actuator DM, updated at a desired rate of 250 Hz.

The real-time control requirement for this system is demanding, though each correction arm is decoupled, and hence can be treated as a separate AO system. Therefore, each of these systems will have 11 84 × 84 WFSs, and one 85 × 85 actuator DM, leading to a total of about 125 000 slope measurements and 5800 active actuators. A control matrix for this system would have a size of nearly 3 GB (assuming that elements are stored as a 32-bit floating point). Generally, for a system of this size, processing power is not the limiting factor. Rather, it is the memory bandwidth required to load this control matrix from memory into the processing units for each frame (be they CPUs, GPUs, FPGAs, etc.), which in this case is equal to about 725 GB s^{-1} (matrix size multiplied by frame rate). The three-GPU system that we have in Durham has a peak theoretical bandwidth of 444 GB s^{-1}, and our matrix–vector multiplication core is able to reach about 70 per cent of this. Theoretical or even measured matrix multiplication rates are however not a good benchmark for a RTCS. This is because the RTCS will perform additional operations, which will affect cache or resource usage, and furthermore, the multiplication will be broken up into blocks allowing reconstruction to be interspersed with pixel read-out. DARC is an ideal tool for such benchmarking, as it is both a full RTCS, but also flexible enough to investigate parameters for optimal performance.

By using the three Tesla GPUs that we have available, we are able to process one WFS on each GPU at a frame rate of about 300 Hz, with wavefront reconstruction for one correction arm (i.e. 3/11th of a single channel). We find that the frame rate falls slightly with the number of WFSs processed (and hence the number of GPUs used) as shown in Fig. 7. If we consider how a system containing eight GPUs might behave (the maximum number of GPUs that can be placed in a single PC), then an extrapolation to eight WFSs and GPUs (which we realize is rather dubious from the available data, but will suffice for the argument being made here) might bring the frame rate down to about 280 Hz. For a single channel of EAGLE, reconstruction from 11 WFSs is required, which if divided between eight GPUs would reduce the frame rate to about 200 Hz. If we were to change the GPUs used from Tesla 2070 cards to more powerful (yet cheaper) GeForce 580 cards (increasing the GPU

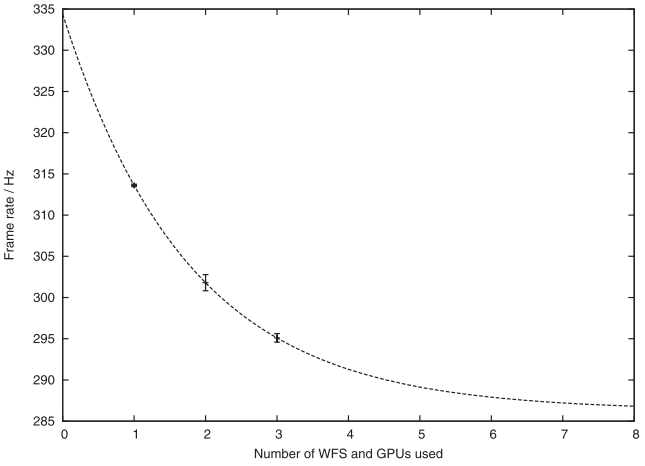

Figure 7. A plot showing how frame rate is affected by the number of WFSs processed for an EAGLE-like system, assuming a GPU dedicated to each WFS all on the same computer. An exponential trend line has been fitted to the data.

internal memory bandwidth from 148 to 192 GB s^{-1}), the frame rate could be increased to 260 Hz.

Since EAGLE channels are essentially independent, we could then replicate this system using a single computer and eight GPUs for each channel (and one more to control the E-ELT deformable M4 mirror with 85 × 85 actuators), giving a full real-time solution for EAGLE. In fact, this situation is even simpler, since we only need to perform wavefront slope measurement once per frame, rather than once per channel per frame, and so a front end (possibly FPGA based for minimal latency, e.g. the SPARTA wavefront processing unit) could be used to compute wavefront slopes, which would greatly reduce the CPU processing power required for each channel (though GPU processing requirement would remain unchanged).

It should be noted that in the case of wavefront reconstruction only (assuming slope calculation is carried out elsewhere), our system in Durham is able to achieve a frame rate of 400 Hz, independent of whether we process slopes from one, two or three WFSs using a GPU for each. We are therefore confident that such a system would allow us to implement the entire EAGLE RTCS using currently available hardware and software. Given the performance improvements promised in both multicore CPUs and in GPUs over the next few years, a CPU- and GPU-based solution for EAGLE is even more feasible.

4.4 Future real-time control requirements

In addition to the consideration of real-time control for EAGLE given in Section 4.3, we should also consider other future real-time requirements, whether DARC will be able to meet these, and what the AO community will require.

The most demanding currently proposed instrument (in terms of computational requirement) is probably Exo-Planet Imaging Camera and Spectrograph (EPICS; Kasper et al. 2007) for the E-ELT. This XAO system consists of a WFS with 200 × 200 subapertures and a frame rate of 2 kHz. Using DARC with currently available GPUs (NVIDIA GeForce 580 with a memory bandwidth of 192 GB s^{-1}), performing matrix–vector multiplication-based wavefront reconstruction would require a system with at least 150 GPUs. Although this is a similar number to that required by EAGLE, for EPICS the results from each GPU must be combined with results from all other GPUs, and to the authors, this does not seem to be a practical solution when a frame rate of 2 kHz is required (for EAGLE, only computations from sets of eight GPUs need to be combined since the MOAO arms can be treated independently). On this scale, other reconstruction algorithms do not seem appropriate: conjugate gradient algorithms cannot be massively parallelized to this degree, and, as far as we are aware, algorithms such as CURE (Rosensteiner 2011) do not yet provide the correction accuracy required. Therefore, we must conclude that DARC is not suitable for this application. However, given that this instrument is at least 10 years away, more powerful hardware and more suitable algorithms are likely to become available.

To our knowledge, other proposed instruments generally have lower computational and memory bandwidth requirements than EAGLE, which we have shown that DARC would be capable of controlling. Therefore, we are confident that DARC provides a real-time control solution for most proposed AO systems, and this demonstrates the suitability of CPU-based RTCSs for AO. It should be noted that it is only within the last few years, with the advent of multicore CPUs and more recently GPU acceleration, that such systems have become feasible, giving the advantages of both greater processing power and additional CPUs to handle non-real-time processes (such as operating system services and diagnostic systems), thus keeping jitter to a minimum.

4.5 Wavefront sensor camera and deformable mirror specifications

The timing measurements for DARC provided here are optimistic since we do not include a physical WFS camera or DM. However, by considering the latency and frame-rate requirements for the systems for which we have investigated the performance of DARC, we can derive the specifications required for these hardware components that will allow the system to perform as desired.

A frame computation time of 323 μs was measured for DARC operating in a VLT planet-finder configuration. The total latency for this system (including camera read-out) must be below 1 ms. However, when camera read-out time and DM settling time are taken into account, this corresponds to an acceptable RTCS latency (from the last pixel received to the last DM command leaving) of about 100 μs (E. Fedrigo, private communication). To achieve this latency, we can therefore specify that the camera pixel stream must be accessible in blocks that are equal to or less than quarter of an image in size (the time to process each block will then be about 80 μs, meaning that it will take this long to finish computation once the last block arrives at the computer, resulting in a 80 μs RTCS latency.

The DARC configuration for a system resembling Palm-3000 provides a processing time of 500 μs using three GPUs. Once a real WFS camera and DM are added, this processing time would increase meaning that the 2 kHz frame rate could no longer be met. Therefore, a fourth GPU is required to be added to the system, which would result in a processing time of less than 400 μs. This therefore provides a 100-μs overhead to allow for the process of sending commands to the DM and obtaining pixel data from a frame grabber card, which should be more than ample (typically this will just be a command to initiate a DMA), thus putting a performance requirement on the hardware. The maximum camera frame rate is 2019 Hz, corresponding to a read-out time of 495 μs. This is greater than the processing time, meaning that we are therefore able to interleave processing with read-out, with enough time to finish block processing between each block of pixels arriving. The RTCS

latency (the last pixel received to the last DM command sent) will therefore be determined by the pixel block size. If we read the camera in four blocks (giving a 100 μs processing time per block) then after the last pixel arrives, we will have a 100 μs processing time plus the 100-μs overhead that we have allowed for data transfer, a total of 200-μs RTCS latency. For this system, we therefore require that the camera pixels can be accessed in four blocks, and that the time overhead for transferring pixels into memory and commanding the DM is less than 100 μs.

Our configuration of DARC for an EAGLE-like instrument gives a processing time of 3.8 ms. To achieve the desired frame rate of 250 Hz, the total processing time must remain below 4 ms. We therefore have 200 μs in which to process the last block of pixels (processing of other blocks will be interleaved with pixel read-out and thus will not contribute to the RTCS latency), and to send commands to the DM. If we divide the pixel data for each camera into 42 blocks (which corresponds to two rows of subapertures per block), then the processing time for each block is 90 μs, thus leaving 110 μs spare. We have therefore placed requirements on the DM and WFS camera for this system: we require a camera that will allow us to access the pixel stream in blocks of size 1/42 of a frame. We also require the time to receive this block of pixel data into computer memory, and to send DM demands from computer memory to be less than 100 μs. This equates to a data bandwidth of about 4 Gbits s^{-1}, which is achievable with a single PCI-Express (generation 2) lane. This requirement can easily be met given that camera interface cards typically use multiple lanes, and a DM interface card could also use multiple lanes.

5 CONCLUSION

We have presented details of the Durham AO RTCS, including recent improvements. We have described some of the more advanced features and algorithms available with DARC and given details about the improved modularity and flexibility of the system (including the ability to load and unload modules while in operation). We have also discussed ways in which DARC can be optimized for specific AO systems.

We have discussed how DARC can be used for almost all currently proposed future AO systems, and given performance estimates for some of these. We are well aware that without a physical camera and DM, the results that we have presented here do not represent the whole system, and so we have used these performance estimates to derive the WFS camera and DM requirements that are required to ensure that an AO system can meet its design performance. The architecture of DARC, allowing interleaving of processing and camera read-out, means that AO system latency can be kept low, and so we are confident that DARC presents a real-time control solution that is well suited to ELT scale systems.

By making use of GPU technology, we have been able to provide a RTCS suited for all but the most ambitious of proposed ELT instruments, and have demonstrated that software and CPU-based RTCSs have now come of age.

ACKNOWLEDGMENTS

This work is funded by the UK Science and Technology Facilities Council. The authors would like to thank the referees who helped to improve this paper.

REFERENCES

Babcock H. W., 1953, PASP, 65, 229
Basden A. G., Geng D., Guzman D., Morris T., Myers R., Saunter S., 2007, Appl. Opt., 46, 6136
Basden A., Geng D., Myers R., Younger E., 2010a, Appl. Opt., 49, 6354
Basden A., Myers R., Butterley T., 2010b, Appl. Opt., 49, G1
Basden A. G., Myers R. M., Gendron E., 2011, MNRAS, 419, 1744
Béchet C., Tallon M., Thiébaut E., 2006, in Hubin N., Max C. E., Wizinowich P. L., eds, Proc. SPIE Vol. 6272, Adaptive Optics Systems. SPIE, Bellingham, p. 94
Correia C., Conan J.-M., Kulcsár C., Raynaud H.-F., Petit C., 2010, in Clénet Y., Conan J.-M., Fusco Th, Rousset G., eds, Adaptive Optics for Extremely Large Telescopes, Vol. 1. EDP Sciences, Paris, p. 7003
Cuby J.-G. et al., 2008, in McLean I. S., Casali M. M., eds, Proc. SPIE Vol. 7014, Ground-based and Airborne Instrumentation for Astronomy II. SPIE, Bellingham, p. 53
Fedrigo E., Donaldson R., Soenke C., Myers R., Goodsell S., Geng D., Saunter C., Dipper N., 2006, in Ellerbroek B. L., Bonaccini Calia D., eds, Proc. SPIE Vol. 6272, Advances in Adaptive Optics II. SPIE, Bellingham, p. 31
Fusco T. et al., 2006, Opt. Express, 14, 7515
Gendron E., Brangier M., Chenegros G., Vidal F., Hubert Z., Rousset G., Pouplard F., 2010, in Clénet Y., Conan J.-M., Fusco Th., Rousset G., eds, 1st AO4ELT Conference – Adaptive Optics for Extremely Large Telescopes, Vol. 1. EDP Sciences, Paris, p. 05003
Gendron E., Vidal F., Brangier M., Morris T., Hubert Z., Basden A., Rousset G., Myers R., 2011, A&A, 529, L2
Gilles L., Ellerbroek B. L., Vogel C. R., 2003, Appl. Opt., 42, 5233
Kasper M. et al., 2007, in Kalas P., ed, In the Spirit of Bernard Lyot: The Direct Detection of Planets and Circumstellar Disks in the 21st Century, Vol. 1. Univ. California, Berkeley, CA, p. 35
Paufique J. et al., 2010, in Ellerbroek B. L., Hart M., Hubin N., Wizinowich P. L., eds, Proc. SPIE Vol. 7736, Adaptive Optics Systems II. SPIE, Bellingham, p. 57
Rosensteiner M., 2011, J. Opt. Soc. Am. A, 28, 2132
Rousset G., Fusco T., Assemat F., Gendron E., Morris T., Robert C., Myers R., 2010, in Ellerbroek B. L., Hart M., Hubin N., Wizinowich P. L., eds, Proc. SPIE Vol. 7736, Adaptive Optics Systems II. SPIE, Bellingham, p. 25
Stuik R., Bacon R., Conzelmann R., Delabre B., Fedrigo E., Hubin N., Le Louarn M., Ströbele S., 2006, New Astron. Rev., 49, 618
Truong T. N. et al., 2008, in Hubin N., Max C. E., Wizinowich P. L., eds, Proc. SPIE Vol. 7015, Conference title Adaptive Optics Systems. SPIE, Bellingham, p. 95

This paper has been typeset from a TEX/LATEX file prepared by the author.