Multi-Frame Blind Deconvolution Accelerated with Graphical Processing Units

Michael Werth, Brandoch Calef, Kevin Roe
The Boeing Company
Amanda Conti
Air Force Research Laboratory

ABSTRACT

Ground-based observations of Low Earth Orbit (LEO) satellites are significantly degraded by atmospheric turbulence. Multi-Frame Blind Deconvolution (MFBD) is a class of algorithm that attempts to solve this ill-posed inverse problem using a forward model approach. It assumes that a Point Spread Function (PSF) is convolved with a pristine image of the target and then iteratively solves for both simultaneously. Processing one full LEO collection in this way requires an extraordinary amount of processing power and is a task that is normally performed with supercomputers to minimize processing time. In this paper, we describe a new MFBD implementation written with NVidias Compute Unified Device Architecture (CUDA) language to make use of the incredible parallelization capabilities of graphical processing units (GPUs). This reduces the scope of hardware capable of achieving the same processing time from a supercomputer-scale system to a single desktop or server. Measurements of processing time and image quality for several reference objects are compared against results from the latest version of Physically Constrained Iterative Deconvolution (PCID), a gold-standard MFBD implementation that uses the Message Passing Interface (MPI) to make use of numerous CPUs in high-performance computing environments.

1. INTRODUCTION

The Air Force Maui Optical and Supercomputing (AMOS) site hosts a 1.6-meter and a 3.6-meter telescope on the summit of Mt. Haleakala. At an observing wavelength of 800 nm, the diffraction-limited resolution of the 3.6-m telescope is:

$$\lambda/D = 0.22\ \mu\text{rad} \tag{1}$$

Observing a Low Earth Orbit (LEO) object at a distance of 500 km, and in perfect conditions, this telescope should be able to resolve details as small as 11 cm. However, even Haleakala's exceptionally still atmosphere imposes significant blurring due to turbulence; with a median Fried's parameter (r_0) value of 10 cm [1], the effective resolution is only:

$$\lambda/r_0 = 8.0\ \mu\text{rad} \tag{2}$$

This limits the spatial resolution to 4 meters while observing an object 500 km away. The random nature of turbulence may permit using lucky imaging to sometimes resolve fine details, but Fried [2] suggests that the probability of getting a good image for a 3.6-m diameter aperture at r_0 =10 cm is:

$$5.6e^{[0.1557(D/r_0)^2]} = 10^{-87} \tag{3}$$

For comparison, the chance of being struck by lightning in the United States at least once in a human lifetime is about 10^{-6}, and the odds of winning some of the largest lotteries is about 10^{-9}.

Thankfully, numerous techniques have been developed to mitigate the blurring effects of turbulence, including hardware-based solutions such as Adaptive Optics (AO) [3, 4, 5, 6] as well as numerous software-based image recovery algorithms such as bispectrum image recovery [7, 8, 9] and Multi-Frame Blind Deconvolution (MFBD) [10, 11, 12, 13],

among others. The gold-standard image recovery algorithm used at the AMOS site is Physically-Constrained Iterative Deconvolution (PCID) [14, 15, 16, 17], an implementation of MFBD that attempts to simultaneously recover the object and the blur function using iterative maximum likelihood estimation while imposing a number of physical constraints, such as positivity and support constraints. PCID (like most MFBD algorithms) is computationally expensive, relying primarily on Fast Fourier Transforms (FFTs), and producing timely output images necessitates using high-performance computing (HPC) resources dedicated to this task.

Researchers and HPC systems alike have been shifting toward using commonly-available General-Purpose Graphical Processing Unit (GPGPU) hardware for solving problems with a high degree of data parallelization [18] using software architectures such as Open Computing Language (OpenCL) [19] and NVidia's Compute-Unified Device Architecture (CUDA) [20, 21]. It's easy to imagine how hardware designed to operate on image data could be more efficient at running image recovery algorithms than a more generic Central Processing Unit (CPU); any given image recovery technique is likely going to perform a common set of operations on an array of pixel data, or multiple arrays of such data, which is almost an ideal use case for GPGPUs. PCID in particular uses numerous Fast Fourier Transforms (FFTs) during the iterative minimization process, and for normal usage its overall performance is constrained by the rate at which it can perform 2D FFTs. However, PCID only makes use of Central Processing Unit (CPU) hardware with Message Parsing Interface (MPI) [22] for parallelization between multiple CPUs. It uses the very well-written Fastest Fourier Transform in the West (FFTW) [23], but even images as small as 64x64 can be processed more than an order of magnitude faster using the CUDA API's cufft library [24, 25].

In this paper we describe a new MFBD algorithm that is implemented in CUDA. The MFBD algorithm, named Likelihood-based Uncertainty-Constrained Iterative Deconvolution (LUCID), will be described in Section 2. CUDA-specific implementation details are described in Section 3. Sections 4 describes the performance testing configuration and Section 5 describes the performance measurements for the LUCID and PCID algorithms. This paper will conclude with a summary of the results and potential improvements in Section 6.

2. LUCID

The LUCID algorithm is fundamentally based on the PCID algorithm. It assumes a simple linear forward model for imaging through turbulence. Each mth measurement of short-exposure image data, $d_m(\mathbf{x})$, is represented by a pristine image of the object, $o(\mathbf{x})$, convolved with a Point Spread Function (PSF), $h_m(\mathbf{x})$, plus some additive noise $n_m(\mathbf{x})$, as shown in the Equation 4:

$$d_m(\mathbf{x}) = o(\mathbf{x}) * h_m(\mathbf{x}) + n_m(\mathbf{x}) \tag{4}$$

Equation 4 assumes that the pristine object $o(\mathbf{x})$ is unchanging across all m measurement frames. As a practical concern, this equation remains approximately true for only a few seconds at a time when observing a LEO satellite from a fixed ground site due to naturally-occurring changes in the target's pose. If the satellite is spinning or possess moving components, then the length of time for which this equation remains approximately true is further constrained. Thus, processing a full pass entails grouping collected images into a series of short ensembles that are processed separately. For each of these image ensembles, the object and PSFs are estimated simultaneously in the image domain by minimizing the following cost function:

$$J[o(\mathbf{x}), h_1(\mathbf{x}), ..., h_m(\mathbf{x})] = \sum_{m=1}^{M} \sum_{n=1}^{N} \frac{(i_m(\mathbf{x}_n) - d_m(\mathbf{x}_n))^2}{\sigma^2(\mathbf{x}_n) + i_m(\mathbf{x}_n)} \tag{5}$$

Equation 5 is the sum over M frames and N pixels per frame of the noise-normalized squared residual between the measured data frames $d(\mathbf{x}_n)$ and the data model $i(\mathbf{x}_n)$. The denominator contains read noise variance per-pixel of $\sigma^2(\mathbf{x}_n)$ and shot noise variance of $i_m(\mathbf{x}_n)$. Each mth model of the data is calculated from the convolution of the estimated object frame with each estimated PSF:

$$i_m(\mathbf{x}_n) = o(\mathbf{x}) * h_m(\mathbf{x}_n) \tag{6}$$

New object and PSF images are estimated using the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS)

quasi-Newton method [26, 27] with the cost function defined in Equation 5. The data model frames are regularized using the filter-based regularization scheme used by the PCID algorithm [16], wherein $i_m(\mathbf{x}_n)$ is convolved with a 2D filter so as to achieve a reduction in noise amplification. However, noise-amplification is still inescapable, so output images are saved as a function of iteration number. The best image in each ensemble is chosen later from an image quality assessment performed on the full range of the ensemble's output images.

3. CUDA

Using CUDA to implement a MFBD algorithm introduces a number of specific concerns that do not exist in a CPU implementation. First, it is generally understood that GPU processing suffers from significant overhead in moving data to and from the GPU device [21]. Thus, the software moves a full ensemble of image data to the device once and then perform all subsequent processing on the GPU. Figure 1 demonstrates this; the partitioning of a collection into short ensembles is performed by the CPU, and then all of the data is sent to and remains on the GPU until processing is complete.

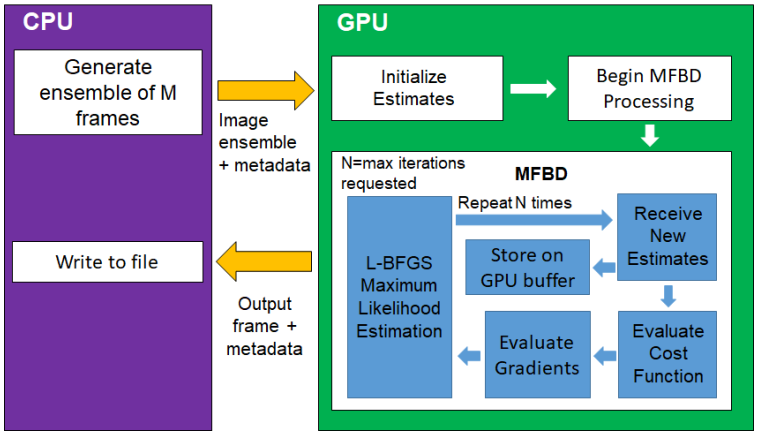


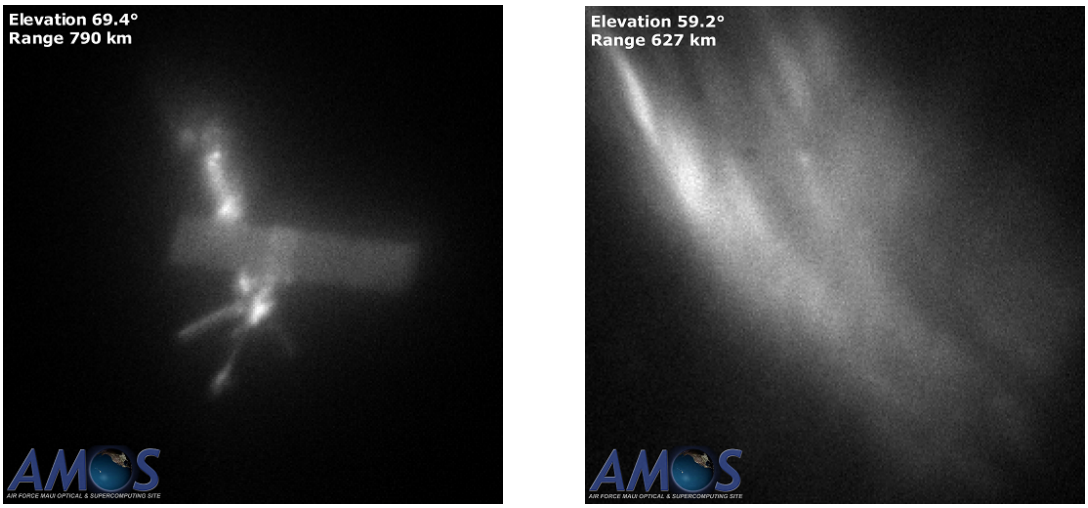
Fig. 1: Model of how data is transferred between host (CPU) and device (GPU) memory during LUCID execution

Consumer-grade GPUs, such as NVidia's GTX and RTX series, are optimized for single-precision (float32) computation; while they are still capable of performing double-precision (float64) calculations, the performance on these devices suffers by $1/32$, as shown in the Arithmetic Instructions section of the CUDA Programming Guide [21]. For CPUs, and for NVidia's datacenter-grade Tesla line, the performance scaling is normally $1/2$. The additional precision available to float64 has no impact on LUCID's (or PCID's) ability to reconstruct images during the minimization process, which leads to a design that uses 32-bit precision wherever possible.

4. PERFORMANCE TEST SETUP

Images of Seasat and Hubble Space Telescope (HST) were collected with a EMCCD camera on the 3.6-meter Advanced Electro-Optical System (AEOS) Telescope. All images were collected with an exposure time of 15 ms and in high-gain EMCCD mode for a collection of at least 10,000 frames (75 seconds) per target. Examples of these images are shown in Figure 2.

For a collection with a fixed number of ensembles, the total processing time is primarily dependent on the size of each ensemble and the number of processing iterations. Thus, these are the parameters varied for a processing time comparison between LUCID and PCID. These algorithms both use the same cost function (Equation 5) and the same minimizer (L-BFGS [26, 27]), but PCID runs on CPU cores whereas LUCID uses a GPU. Each satellite collection is reduced to a 10,000 frame dataset and then split up into 150 separate ensembles, where the start point of each ensemble is separated by 1 second (67 frames). Total processing time is measured as a function of ensemble size, which is varied between 10 and 300 frames while performing 50 MFBD iterations per ensemble, and as a function of number of MFBD iterations per ensemble, which is varied between 10 and 300 with an ensemble size of 50 frames.



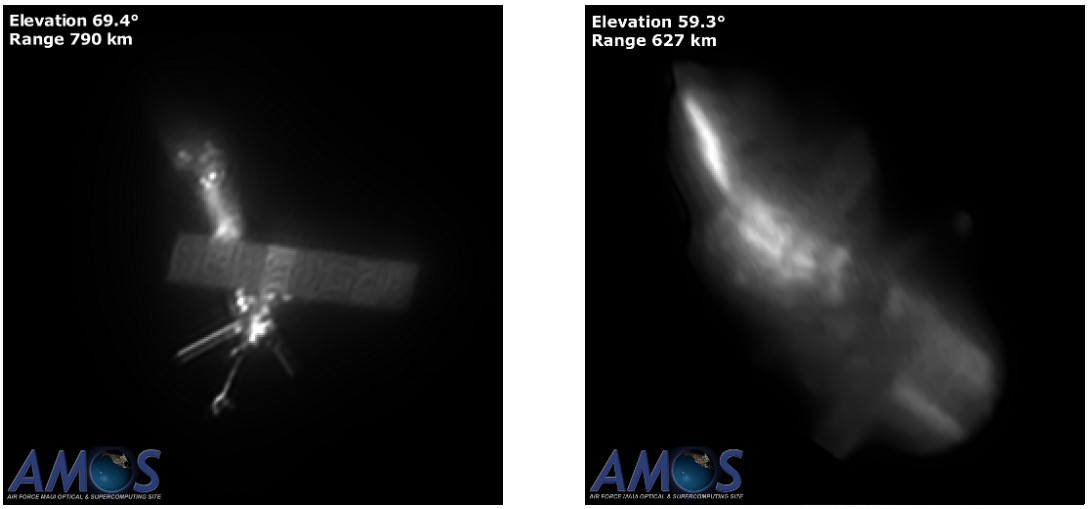
(a) (b)

Fig. 2: Sample input images for (a) Seasat and (b) HST

Processing time is measured on a handful of hardware platforms. The PCID algorithm executed on a system with a Intel Xeon E5-2698v4 20-core Broadwell CPU using all available cores. LUCID was executed on two different GPUs in separate benchmarking experiments: an NVidia GTX 980 and an NVidia Tesla V100. Each benchmarking experiment is granted exclusive use of the hardware so as to minimize uncertainty. Benchmarking measurements are made using `Python`'s `timeit` module to measure the full execution time of the process wrapped in a `subprocess.Popen` instance. This includes the time to read image data from disk, write data to disk, and (in the case of LUCID only) transfer data from the host to the GPU device. Each experiment is executed 10 times, and then 95% confidence intervals are estimated.

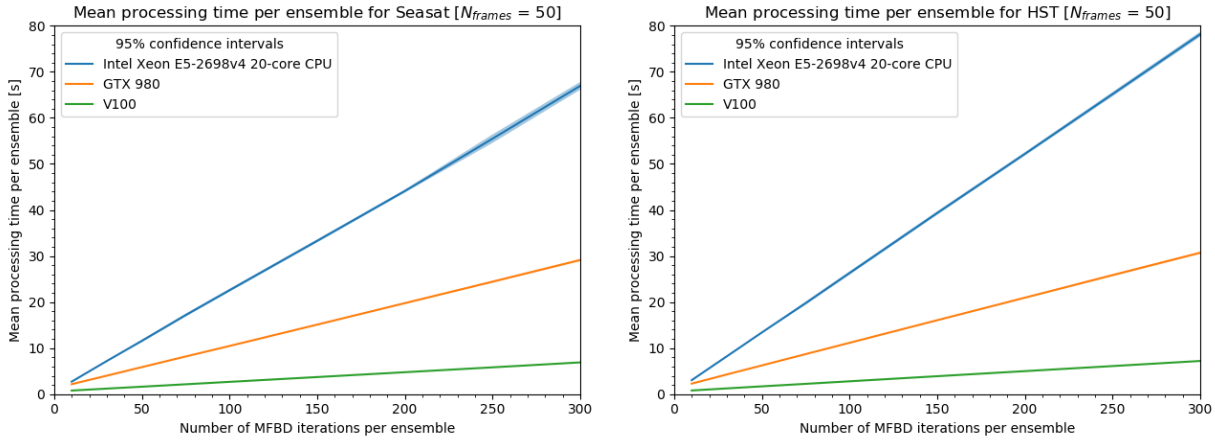
5. PERFORMANCE MEASUREMENTS

The LUCID and PCID algorithms produce images of equivalent quality. Examples of LUCID output images are shown in Figure 3, but both algorithms produced images with the same image quality.



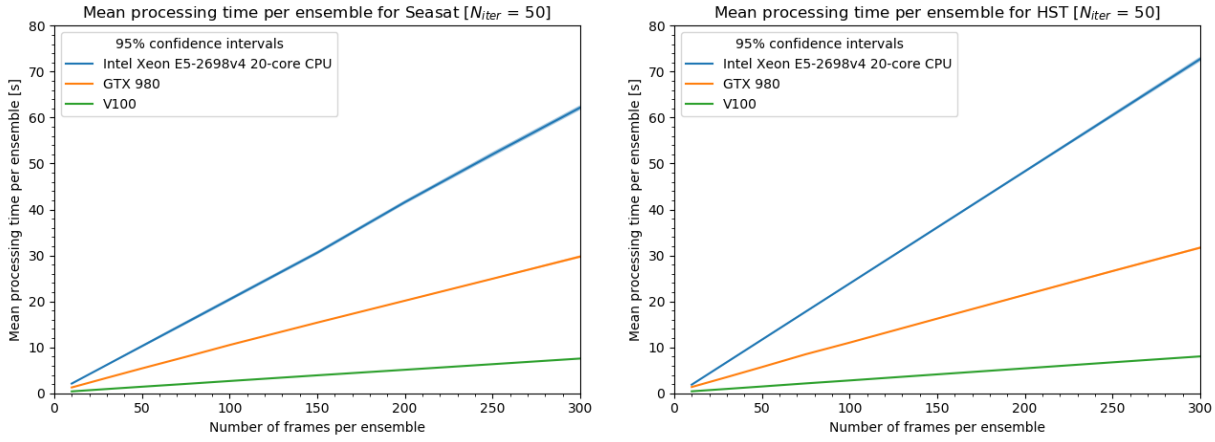
(a) (b)

Fig. 3: Sample output images for (a) Seasat and (b) HST



(a) (b)

Fig. 4: Processing time versus number of MFBD iterations per ensemble for several hardware platforms when processing (a) Seasat and (b) HST



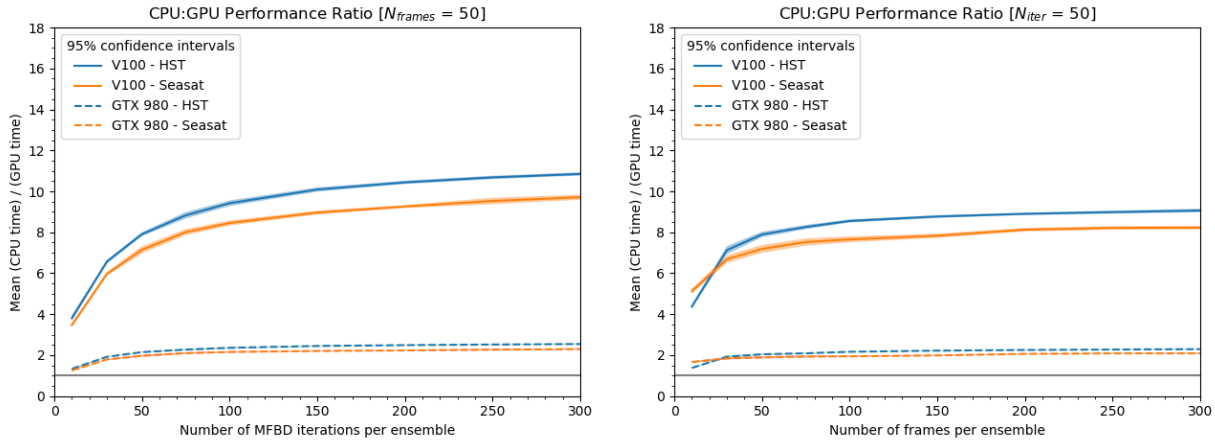
(a) (b)

Fig. 5: Processing time versus number of frames per ensemble for several hardware platforms when processing (a) Seasat and (b) HST

Figure 4 shows the average processing time per ensemble as a function of iteration number and Figure 5 shows the average processing time per ensemble as a function of ensemble size for each of the satellites considered in this paper. Each data point in these figures reports the mean and standard deviation of 10 benchmark experiments. Figure 6 shows the ratio of LUCID processing time to PCID processing time for each of the objects and varied parameters.

6. CONCLUSIONS AND FUTURE WORK

LUCID is an application written in C++ and CUDA that uses GPU hardware to quickly process degraded images of LEO satellites using a MFBD implementation. Figure 6 shows that LUCID running on a single Tesla V100 with 100 MFBD iterations can process data up to 10 times faster than PCID using all 20 cores of a recent-generation CPU. The shapes of these curves demonstrate the presence of overhead cost when using the GPU for very small problem sizes (10 iterations or 10 frames per ensemble), and that this performance cost becomes negligible for larger problem sizes. However, LUCID still runs 4 times faster even in the worst case on the V100. This suggests that LUCID has supercomputer-equivalent capabilities to PCID, especially on systems that may have multiple V100 GPUs



(a) (b)

Fig. 6: 95% confidence intervals for the ratio of PCID (CPU) processing time per ensemble to LUCID (GPU) processing time per ensemble when varying (a) the number of iterations and (b) the number of frames per ensemble. A grey line is drawn at a ratio of 1 for visual reference, as any data points greater than 1 illustrate GPU performance exceeding CPU performance.

per computational node, representing a significant reduction in hardware procurement and sustainment costs. The difference in performance ratio between Seasat and HST data shown in Figure 6 is well-explained by Seasat data being present in about 10% more of the camera pixels on average than the HST data, by virtue of some combination of differences in object size and PSF size.

The performance improvement when running LUCID with a GTX 980 is a little less promising, offering about 2 times more speed than PCID. However, the economics of this comparison paint a considerably brighter picture. First, the GTX 980 is considerably older hardware than the Intel Xeon E5-2698v4 against which it is being compared; the GTX 980 GPU was released in 2014 [28] whereas the Xeon used in this study was released in 2016 [29]. The GPU was also about 6 times cheaper on its launch date than the CPU was on its launch date according to each manufacturer's suggested retail price (MSRP). With that in mind, future LUCID benchmarking measurements should include recently released consumer-grade GPU hardware, such as the GTX 2080. The advantage of the GPU acting as a coprocessor should also be mentioned; LUCID's performance does not suffer when other CPU-using applications are being used on the same system.

Future efforts will include refinement of the algorithm to further improve processing time as well as image quality. The current prototype has numerous avenues for potential improvements that should reduce the processing time, improve the image quality, or both. For instance, much of the LUCID source code reuses device memory from already-processed ensembles so as to reduce superfluous (and expensive) memory allocation, but the L-BFGS implementation reallocates memory for each new ensemble; the processing time of the algorithm could be improved substantially by providing static memory allocations to L-BFGS for subsequent ensembles. The implementation also uses a single CUDA stream, but multiple streams could be used so as to hide some of the overhead associated with moving data to and from the device [21]. There may be additional avenues for processing time improvements that can be unveiled during future profiling efforts. We are also planning to implement an iteratively reweighted MFBD feature that may provide numerous benefits to image quality [30].

7. REFERENCES

[1] Lawrence William Bradford. Maui4: a 24 hour haleakala turbulence profile. In *Advanced Maui Optical and Space Surveillance Technologies Conference*, Sep 2010.

[2] David L. Fried. Probability of getting a lucky short-exposure image through turbulence. *J. Opt. Soc. Am.*, 68(12):1651–1658, Dec 1978.

[3] Robert K. Tyson. *Principles of Adaptive Optics*. Series in Optics and Optoelectronics. CRC Press, 2010.
[4] Robert K. Tyson and Benjamin W. Frazier. *Field Guide to Adaptive Optics, Second Edition*. Field Guide Series. SPIE, 2012.
[5] John W. Hardy. *Adaptive Optics for Astronomical Telescopes*. Oxford University Press, July 1998.
[6] N. Hubin and L. Noethe. Active optics, adaptive optics, and laser guide stars. *Science*, 262(5138):1390–1394, 1993.
[7] Michael C. Roggeman and Byron M. Welsh. *Imaging Through Turbulence, Second Edition*. CRC Press, Mar 1996.
[8] Pedro Negrete-Regagnon. Practical aspects of image recovery by means of the bispectrum. *J. Opt. Soc. Am. A*, 13(7):1557–1576, Jul 1996.
[9] Charles L. Matson, Tom Soo Hoo, Maria Murphy, Brandoch Calef, Charles C. Beckner, Shiho You, Ron Viloria, and Kathy Borelli. PCID and ASPIRE 2.0 - The Next Generation of AMOS Image Processing Software. *AMOS Conference Technical Proceedings*, page E60, 2007.
[10] Timothy J. Schulz. Multiframe blind deconvolution of astronomical images. *J. Opt. Soc. Am. A*, 10(5):1064–1073, May 1993.
[11] David Gerwe, Jim Stone, Carlos Luna, and Brandoch Calef. Comparison of maximum-likelihood image and wavefront reconstruction using conventional image, phase diversity, and lenslet diversity data. In *Unconventional Imaging II Proceedings*, volume 6307, Sep 2006.
[12] Hirsch, M., Harmeling, S., Sra, S., and Schölkopf, B. Online multi-frame blind deconvolution with super-resolution and saturation correction. *A&A*, 531:A9, Feb 2011.
[13] Michael Hart, Stuart Jefferies, Douglas Hope, E. Keith Hege, Runa Briguglio, Enrico Pinna, Alfio Puglisi, Fernando Quiros, and Marco Xompero. Multiframe blind deconvolution for imaging in daylight and strong turbulence conditions. In *Unconventional Imaging, Wavefront Sensing, and Adaptive Coded Aperture Imaging and Non-Imaging Sensor Systems Proceedings*, volume 8165, 2011.
[14] Stuart M. Jefferies and J.C. Christou. Restoration of astronomical images by iterative blind deconvolution. *Astrophysical Journal*, 415:862–874, October 1993.
[15] Julian C. Christou, E. Keith Hege, Stuart M. Jefferies, and Matthew Cheselka. Physically constrained iterative deconvolution of adaptive optics images. In *Proceedings Volume 3494, Atmospheric Propagation, Adaptive Systems, and Lidar Techniques for Remote Sensing II*, volume 3494, 1998.
[16] Charles L. Matson, Kathy Borelli, Stuart Jefferies, Charles C. Beckner Jr., E. Keith Hege, and Michael Lloyd-Hart. Fast and optimal multiframe blind deconvolution algorithm for high-resolution ground-based imaging of space objects. *Appl. Opt.*, 48(1):A75–A92, Jan 2009.
[17] Michael Werth, Brandoch Calef, Daniel Thompson, Kathy Borelli, and Lisa Thompson. Recent improvements in advanced automated post-processing at the amos observatories. In *IEEE Aerospace Conference*, pages 1–7, March 2015.
[18] Michael Feldman. New gpu-accelerated supercomputers change the balance of power on the top500. *Top500*, Jun 2018.
[19] J. E. Stone, D. Gohara, and G. Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in Science Engineering*, 12(3):66–73, May 2010.
[20] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, March 2008.
[21] NVIDIA Corporation. Cuda c programming guide v10.1.168, May 2019.
[22] Message Passing Interface Forum. Mpi: A message-passing interface standard, version 3.1. Technical report, High-Performance Computing Center Stuttgart, University of Stuttgart, Jun 2015.
[23] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".
[24] B. Cloutier, B. K. Muite, and P. Rigge. Performance of fortran and c gpu extensions for a benchmark suite of fourier pseudospectral algorithms. In *2012 Symposium on Application Accelerators in High Performance Computing*, pages 145–148, July 2012.
[25] Peter Steinbach and Matthias Werner. gearshifft – the fft benchmark suite for heterogeneous platforms. In Julian M. Kunkel, Rio Yokota, Pavan Balaji, and David Keyes, editors, *High Performance Computing*, pages 199–216, Cham, 2017. Springer International Publishing.
[26] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, 1995.

[27] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, December 1997.
[28] Ryan Smith. The nvidia geforce gtx 980 review: Maxwell mark 2, Sep 2014.
[29] Intel xeon processor e5-2698 v4 (50m cache, 2.20 ghz) product specifications, 2016.
[30] Brandoch Calef. Iteratively reweighted blind deconvolution. In *AMOS Conference*, September 2013.