論文 / 著書情報
Article / Book Information

Title	A GPU-accelerated image reduction pipeline
Author	Masafumi Niwano, Katsuhiro L. Murata, Ryo Adachi, Sili Wang, Yutaro Tachibana, Yoichi Yatsu, Nobuyuki Kawai, Takashi Shimokawabe, Ryosuke Itoh
Journal/Book name	Publications of the Astronomical Society of Japan, Volume 73, Issue 1, pp. 14-24
Issue date	2020, 10
Note	This is a pre-copyedited, author-produced version of an article accepted for publication in Publications of the Astronomical Society of Japan following peer review. The version of record Publications of the Astronomical Society of Japan, Volume 73, Issue 1, pp. 14-24 is available online at: https://doi.org/10.1093/pasj/psaa091.

Publ. Astron. Soc. Japan (2018) 00(0), 1–10
doi: 10.1093/pasj/xxx000

GPU-accelerated Image Reduction Pipeline

Masafumi NIWANO[1], Katsuhiro L. MURATA[1], Ryo ADACHI[1], Sili WANG[1], Yutaro TACHIBANA[1], Yoichi YATSU[1], Nobuyuki KAWAI[1], Takashi SHIMOKAWABE[2] and Ryosuke ITOH[1,3]

[1]Department of Physics, Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8551, Japan
[2]Supercomputing Research Division, Information Technology Center, The University of Tokyo, 2-11-16 Yayoi, Bunkyo-ku, Tokyo 113-8658, Japan
[3]Bisei Astronomical Observatory, 1723-70 Ookura, Bisei-cho, Ibara, Okayama 714-1411, Japan

*E-mail: niwano@hp.phys.titech.ac.jp

Received 2020 June 26; Accepted 2020 August 26

Abstract

We developed a high-speed image reduction pipeline using Graphics Processing Units (GPUs) as hardware accelerators. Astronomers desire detecting EM counterpart of gravitational-wave sources as soon as possible for sharing positional information to organize systematic follow-up observations. Therefore, high-speed image processing is important. We developed a new image reduction pipeline for our robotic telescope system, which uses a GPU via a Python package CuPy to achieve high-speed image processing. As a result, the processing speed was increased by more than a factor of forty to that of the current pipeline, while maintaining the same functions.

Key words: techniques: image processing, gravitational waves, gamma-ray burst: general

1 Introduction

Gamma-ray bursts (GRB) and gravitational waves (GW) are short-lived ($\lesssim 10^{2-3}$sec) and no one knows when or where they occur, and we call such phenomena as "transients". In order to understand the physical processes of transients, it is essential to perform multi-wavelength and continuous observations for before, during, and for a reasonable length of time after the occurrence. We aim for observing their afterglow and perform the prompt follow-up observations. However, positional information about the GW source obtained from laser interferometers such as LIGO (LIGO Scientific Collaboration et.al. 2015), Virgo (Acernese et.al. 2014) and KAGRA (Akutsu et.al. 2019) has too large of an uncertainty ($\approx 10^{1-3}\mathrm{deg}^2$) to perform systematic follow-up observations. Thus it is necessary to immediately identify the EM counterpart and send more accurate information to the ground telescope network as soon as possible. To realize that, all processes from receiving an alert to finding the counterpart must be performed in a short period ($\lesssim 0.1$days), and therefore, high-speed image processing becomes the key technology.

High-speed image processing is especially important in the era of time-domain astronomy which requires high data-rate observations consisting of a large number of pixels taken at high cadence. Wide-field surveys such as Pan-STARRS (Chambers et.al. 2016), Subaru-HSC (Miyazaki et.al. 2018), ZTF (Bellm et.al. 2019), and Tomo-e Gozen (Sako et.al. 2018) are typical examples. One of the solutions to deal with the high data-rate is CPU-based parallel computing. For example, ZTF obtains very high-resolution CCD images of about 600 Mpix every 40 sec, and its data rate exceeds 200 Mbit/s. To process those extremely fast-growing data, they perform parallel computing in a com-

Table 1. Performance comparison of CPU and GPU

	Name	FP64[1] [TFLOPS]	Power[2] [W]
CPU	Intel Xeon E5-2640v4	0.4	90
GPU	NVIDIA Tesla P100 (PCIe)	4.7	250

[1] Theoretical maximum number of 64-bit floating-point operations performed per second. The CPU value is calculated with IPC=16 (instructions per clock) and Clock=2.4GHz, and the GPU value is nominal.
[2] The CPU value is the thermal design power, and the GPU value is the maximum power consumption.

puting system with 1192 physical CPU cores. As a result, they have achieved the real-time detection of transients and moving objects (Masci et.al. 2019).

A Graphics Processing Unit (GPU) is a hardware-accelerator for graphics processing. A GPU has thousands of computing units called "shaders", which correspond to CPU cores, and the GPU achieves extremely high performance only for SIMD[1] processing (e.g. matrix operation). Although that is achieved by specializing in graphics processing, this performance is sometimes utilized for not only graphics processing, but also other purposes, and such GPU utilization is called GPGPU (General Purpose computing on GPU). In the astronomy, GPGPU is applied for physical simulation, machine learning, analysis of radio interferometer data, to name of few (Tobias et.al. 2009, Tuccillo et.al. 2018, Tazzari et.al. 2018). To build the same-scale computing system as ZTF, 120 Xeon E5-2640v4[2] processors are required and these consume about 10 kW of power. However, with Tesla P100, the same theoretical performance can be achieved with only 10 modules, and the power consumption will be less than 2.5 kW (table 1). That advantage is limited to the case when the performance of the GPU can be fully exploited, but it is not so difficult because image processing is a speciality of the GPU.

The purpose of this research is to utilize GPU for astronomical image processing, and more specifically, to speed up the image reduction pipeline of our robotic telescopes, MITSuME-Akeno and MITSuME-Okayama (Kotani et.al. 2005, Yatsu et.al. 2007, Shimokawabe et.al. 2008).

2 Development

2.1 Image reduction at MITSuME

Both of MITSuME telescopes are equipped with three CCD cameras to obtain simultaneously three-band images. Each CCD generates a 16-bit integer image of 1074×1024 pixels including an overscan area, which is stored as a FITS file. Typical value of exposure time, overhead and data volume is 60 sec (for GW or GRB), 8 sec and 4 GB/night (for 2 telescopes), respectively. To reduce the pixel randomization, we perform 9 points dithering. Therefore positions of point sources on the detector differs for each exposure even observing the same object. After derivation of pedestal level and astrometry, FITS files are transferred to the data server in Tokyo Tech.

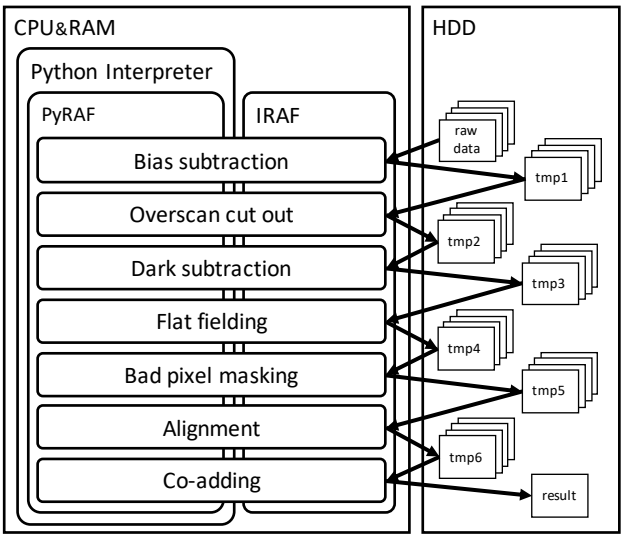

Fig. 1. Current Pipeline overview: Rectangular objects in a frame labeled "HDD" (Hard Disk Drive) indicate FITS files, "raw data" is a set of unreduced data, "result" is a reduced data, and "tmp∗" are sets of a temporary file. Arrows indicate data flows. Only the processing performed by IRAF is shown, and there are actually other processing.

The current image reduction pipeline is implemented as a Python[3] script which uses IRAF[4] (Doug 1986, Doug 1993) via PyRAF[5], and its sequence is shown in figure 1. Specifically, overscan cutout, bias subtraction, dark subtraction, flat fielding, bad pixel masking, alignment, and co-adding are performed by calling IRAF tasks. At the stage of bias subtraction, casting from 16-bit integer to 32-bit floating point is performed. In the overscan cutout, overscans of 50×1024 pix and 2 pix from the edge are removed, leaving the image of 1020×1020 pix. Because of the dithering, alignment is necessary before co-adding. The pipeline uses bicubic-spline interpolation and sigma-clipped-mean algorithm for sub-pixel image shifting and co-adding, respectively. Only the processing performed by IRAF/PyRAF is shown here, and other processing such as calculating the relative position of images using WCSTools[6] is not shown.

[1] Single Instruction Multiple Data
[2] CPU used at ZTF Science Data System
[3] https://www.python.org/
[4] http://ast.noao.edu/data/software
[5] http://www.stsci.edu/institute/software_hardware/pyraf
[6] http://tdc-www.harvard.edu/wcstools/

2.2 New Reduction Pipeline

We created a new pipeline by using Python, because Python is an easy-to-code language, and there are Python packages available for the purpose of this research. The new pipeline uses NumPy[7] (Walt et.al. 2011), Astropy[8] (Astropy Collaboration et.al. 2018), PyWCS[9] and CuPy[10]. CuPy is a Python package for numerical calculation on GPU using CUDA[11]. The most remarkable feature of CuPy is the API compatibility with NumPy, thus a code written with NumPy can be easily rewritten with CuPy.

The sequence of the new pipeline is described in figure 2. Specifically, the header and image data are read from FITS file as instances of `astropy.io.fits.Header`[12] and `numpy.ndarray`[13]. At the same time, casting from 16-bit integer to 32-bit floating point is also performed. The header is passed to `pywcs.WCS`[14] constructor for calculating the relative position of the images, and the image data is converted to `cupy.ndarray`[15] while slicing overscan. Then, bias subtraction, dark subtraction, flat fielding, bad pixel masking, alignment, and co-adding are performed on the GPU with CuPy. The array of resulting image is converted to `numpy.ndarray`, and then written to a FITS file with Astropy. The pipeline uses bicubic-spline interpolation and sigma-clipped-mean algorithm, the same as the current pipeline. The image processing is performed on the GPU-side because it is basically a SIMD operation, and the other processing is performed on the CPU-side because there are few benefits performing it on the GPU or it cannot be performed on the GPU. The overscan cutout is performed on the latter because the array-slicing operation is not SIMD, and transferring data not used for calculation to the VRAM generates extra memory allocation time. `astropy.wcs.WCS` has the same functions as `pywcs.WCS`, but an initialization of the former takes ≈ 7ms while the latter takes $\approx 200\mu$s in our environment.

3 Verification and Performance Test

We evaluated consistency and performance of the new pipeline. Information about the execution environment is shown in tables 2 and 3.

[7] https://numpy.org/
[8] https://www.astropy.org/index.html
[9] https://pypi.org/project/pywcs/
[10] https://cupy.chainer.org/
[11] https://developer.nvidia.com/cuda-zone
[12] Python class provided by Astropy for storing contents of FITS header
[13] Python class of multidimensional array provided by NumPy
[14] Python class for converting coord between WCS and other system
[15] Python class corresponding to `numpy.ndarray` in CuPy

3.1 Consistency Evaluation

We coded the bicubic-spline algorithm and sigma-clipped-mean algorithm ourselves, because CuPy did not provide corresponding functions. Therefore we had to confirm whether the same result as the current pipeline can be reproduced for these two processing methods. We defined $\Delta\hat{I}$ as a quantity for evaluating relative difference between two images as follows,

$$\Delta\hat{I}_{ij} \equiv \frac{I_{ij} - I^{\rm ref}_{ij}}{I^{\rm ref}_{ij}} \tag{1}$$

where I_{ij} is the pixel value of the test image, $I^{\rm ref}_{ij}$ is the pixel value of the reference image, and i, j are indices of a pixel in the image.

3.1.1 Shift Consistency

For the image processed up to bad-pixel masking in the reduction, we created the following two sets of images with various shift amounts, and calculated $\Delta\hat{I}$.

A The image shifted by the new pipeline, using a shifted image made by IRAF-`imshift` as a reference.

B The image that was shifted once and then shifted back to the original position by the new pipeline, using the original image as a reference.

A is for evaluating the consistency with IRAF, and B is used to evaluating how much information of the original image is conserved. We defined a coordinate system in which the lower left corner of the image is the origin, the horizontal direction is the X-axis, and the vertical is the Y-axis. Then, the shift amounts in the X, Y dimensions are ΔX, ΔY. The data used was obtained by applying overscan cut out, bias & dark subtraction, flat fielding, and bad-pixel masking to four of the images acquired in MITSuME-Akeno on May 14, 2018. These are shown by the table 4 and figure 3.

Typical results are shown in figure 4 and figure 5 for A, and figure 6 and figure 7 for B. Figure 4 and figure 6 are images in which each pixel is appropriately colored with $\Delta\hat{I}$, and figure 5 and figure 7 are histograms of $\Delta\hat{I}$. The difference from IRAF is about $10^{-6} - 10^{-5}$. On the other hand, the strange pattern seen in figure 4 suggests that this difference is not random but systematic. Specifically, with the boundary X ≈ 240, $|\Delta\hat{I}|$ is small on the left side and large on the right side. Around the locations where the original image has point sources, $\Delta\hat{I}$ is negative at near side from the boundary and positive at far side. The difference from the original image is about $10^{-2} - 10^{-1}$, and no particular pattern as seen in figure 4 appears. The difference from IRAF is 3 to 4 orders of magnitude smaller than that from the original image. Therefore, the effect of the

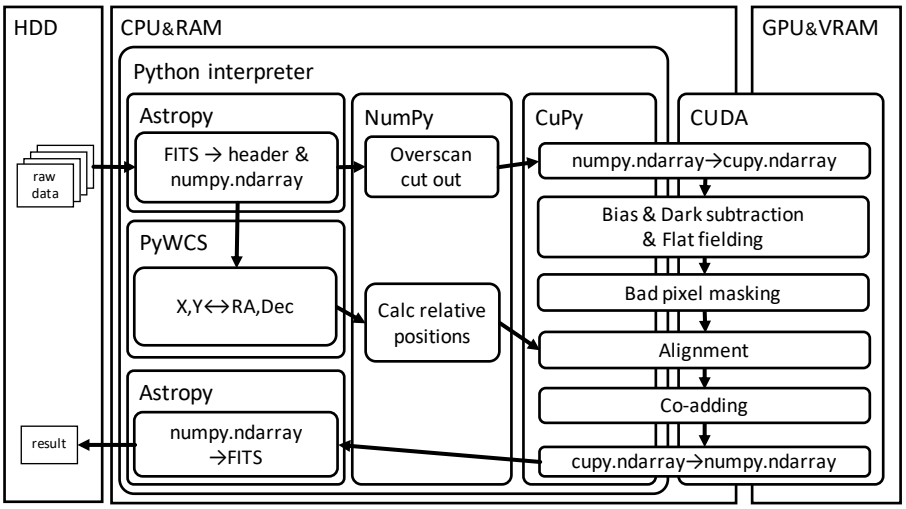

Fig. 2. New pipeline overview: Rectangular objects on "HDD" indicate FITS files, "raw data" is a set of unreduced data, "result" is a reduced data. Arrows indicate data flows. Roughly, "raw data" are read by Astropy, array calculation is performed by CuPy, and the result is written to "result" by Astropy.

Table 2. Machine Specification

	Model	Notes
M/B	Supermicro X10SRA	C612
CPU	Intel Xeon E5-1650 v4	3.6-4.0GHz
RAM	A2ZEON RD4R16G44S2400 ×4	DDR4 2400MHz Registered ECC, Quad Channel
GPU	NVIDIA TITAN X (Pascal)	3584 shaders, 1417-1531MHz, GDDR5X 12GB, PCIe 3.0 x16
SSD	Intel SSDSC2BB12	SATA 6Gbps, OS and executable files are located
HDD	Seagate ST2000VN0001	SATA 6Gbps, Script files and data are located

Table 3. Software versions

OS	Ubuntu 14.04 LTS x86_64		
Python	2.7.6	CUDA	10.0
IRAF	2.16.1	NumPy	1.15.4
PyRAF	2.1.14	Astropy	2.0.9
SExtractor	2.8.6	PyWCS	1.12
WCSTools	3.8.7	CuPy	5.1.0

* Python packages other than PyRAF were installed with `pip`.

Table 4. Information of data used for verification of shift consistency

Label[1]	BG-LEVEL[2] [ADU]	BG-RMS[3] [ADU]
Image1	705	23.741
Image2	197	14.072
Image3	68	13.514
Image4	699	22.267

[1] An appropriate name to identify the data
[2] Median of pixel values
[3] Standard deviation of pixel values
* For all images, the shape is 1020×1020 and the numerical format is a 32-bit float.

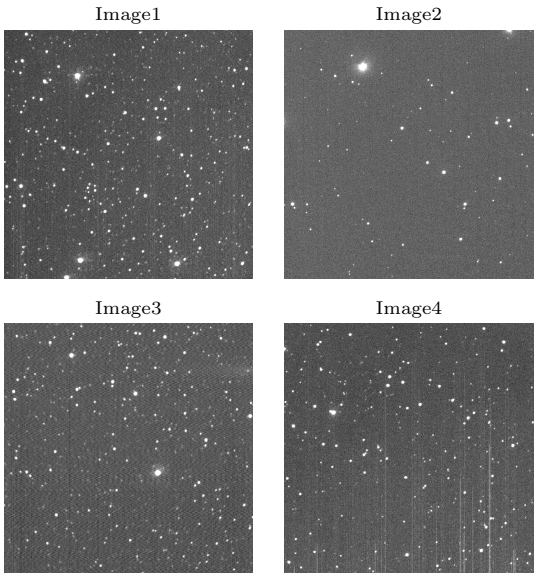

Fig. 3. Data used for verification of shift consistency

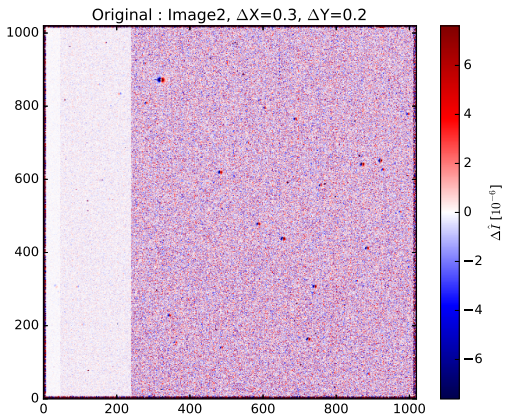

Fig. 4. An image in which each pixel is appropriately colored with $\Delta\hat{I}$ calculated for set A. Image2 is used, and $(\Delta\mathrm{X}, \Delta\mathrm{Y}) = (0.3, 0.2)$. With the boundary $\mathrm{X} \approx 240$, $|\Delta\hat{I}|$ is small on the left side and large on the right side. Around the locations where the original image has point sources, $\Delta\hat{I}$ is negative at near side from the boundary and positive at far side.

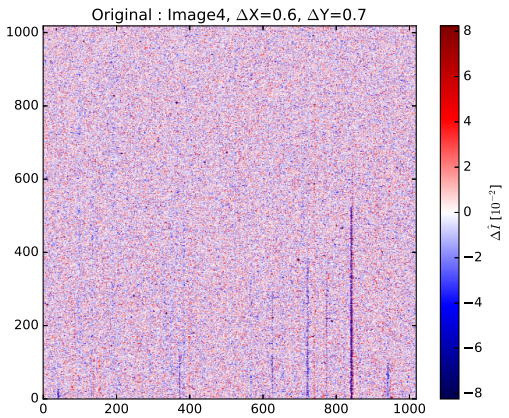

Fig. 6. An image in which each pixel is appropriately colored with $\Delta\hat{I}$ calculated for set B . Image4 is used, and $(\Delta\mathrm{X}, \Delta\mathrm{Y}) = (0.6, 0.7)$. No particular pattern appears other than the pattern around the vertical line in the original image.

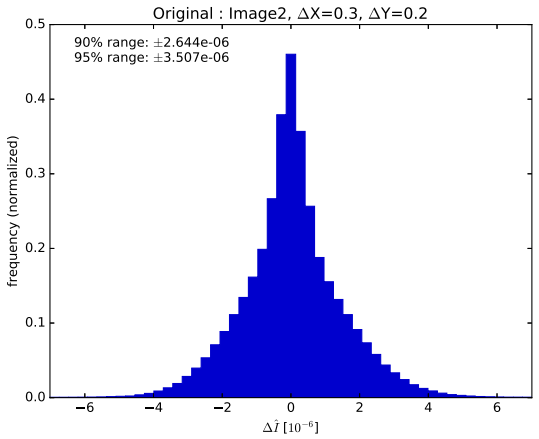

Fig. 5. A histogram of $\Delta\hat{I}$ calculated for set A. Image2 is used, and $(\Delta\mathrm{X}, \Delta\mathrm{Y}) = (0.3, 0.2)$. "90(95)% range" is the range in which 90(95)% of $\Delta\hat{I}$ of all pixels fall. $\Delta\hat{I}$ is about $10^{-6} - 10^{-5}$.

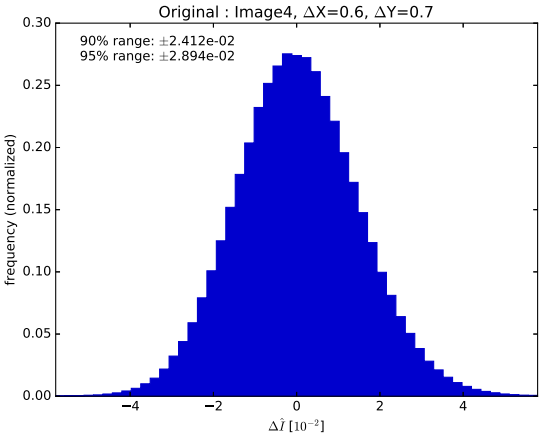

Fig. 7. A histogram of $\Delta\hat{I}$ calculated for set B. Image4 is used, and $(\Delta\mathrm{X}, \Delta\mathrm{Y}) = (0.6, 0.7)$. "90(95)% range" is the range in which 90(95)% of $\Delta\hat{I}$ of all pixels fall. $\Delta\hat{I}$ is about $10^{-2} - 10^{-1}$.

shift operation itself on the result is more dominant than the effect of the change from IRAF to the new pipeline.

We conducted additional experiments to determine the origin of the $10^{-6} - 10^{-5}$ difference. First, we created an image in which two-dimensional cubic B-spline basis functions were arranged (figure 8 and figure 9). Since the cubic B-spline basis function is exactly reproducible by cubic-spline interpolation, a shifted image can be obtained without actually interpolating. Using the shifted image created by this way as a reference image, we obtained $\Delta\hat{I}$ for the image shifted by IRAF-`imshift` and the image shifted by new pipeline. The results are shown in figure 10 and 11. For IRAF-`imshift`, $\Delta\hat{I} \approx 10^{-6}$, and a line of discontinuity as shown in figure 4 appears. On the other hand, for the new pipeline, $\Delta\hat{I} \approx 10^{-7}$, and there is no discontinuity. Therefore, IRAF-`imshift` has a difference of about 10^{-6} from the accurate bicubic-spline interpolation. (It has not been tested in other environments, and it is possible that the bug occurs only in our environment.)

3.1.2 Co-adding Consistency

For the data set processed up to the alignment, we generated a co-added image by a function of the new pipeline, and calculated $\Delta\hat{I}$ using a image co-added by IRAF-`imcombine` as the reference. The input images are acquired at MITSuME-Akeno on May 14, 2018. Information about the input-image sets are given in table 5. The co-added images are shown in figure 12.

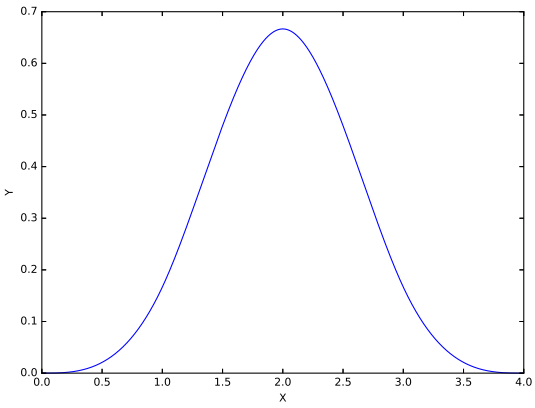

Fig. 8. Cubic B-spline basis function. This function satisfies the cubic-spline constraint conditions, so it can be reproduced exactly with cubic-spline.

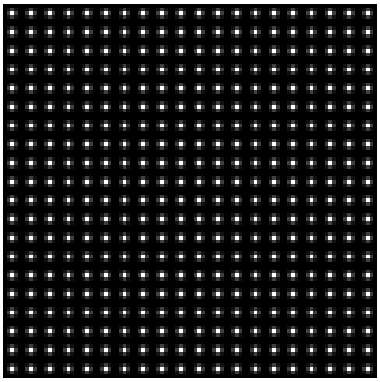

Fig. 9. Two-dimensional cubic B-spline basis functions were arranged in this image. A shifted image of this can be obtained without actually interpolating.

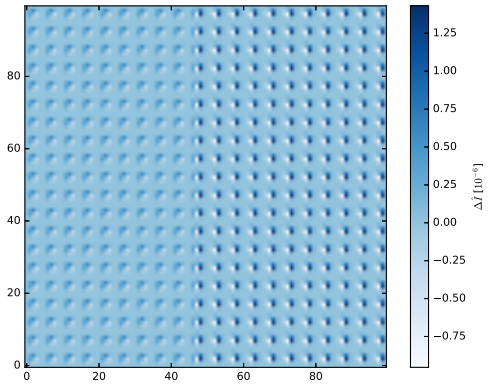

Fig. 10. An image in which each pixel is appropriately colored with $\Delta\hat{I}$ calculated for IRAF shifted image. $(\Delta\mathrm{X}, \Delta\mathrm{Y}) = (0.3, 0.2)$. $\Delta\hat{I}$ is about 10^{-6} and the line of discontinuity as seen in figure 4 appears.

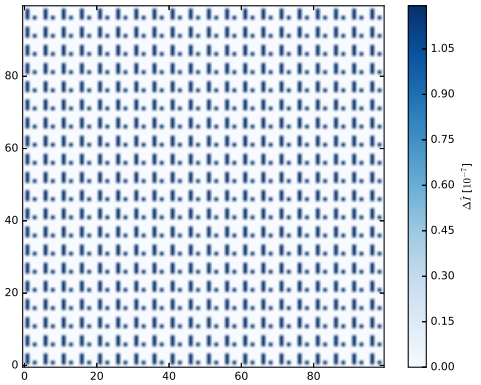

Fig. 11. An image in which each pixel is appropriately colored with $\Delta\hat{I}$ calculated for new pipeline shifted image. $(\Delta\mathrm{X}, \Delta\mathrm{Y}) = (0.3, 0.2)$. $\Delta\hat{I}$ is about 10^{-7} and there is no discontinuity.

Table 5. Information of data-sets used for verification of co-adding consistency

Label[1]	Number of Images	Shape[2] (X×Y) [pix]
Set1	35	910×944
Set2	81	928×864
Set3	97	911×938
Set4	44	811×916

[1] An appropriate name to identify the data-set
[2] The reason why the shape of the image is different from that before the alignment is that an area which does not overlap with another image is cut out during the alignment.
* For all data-set, the numerical format is a 32-bit float.

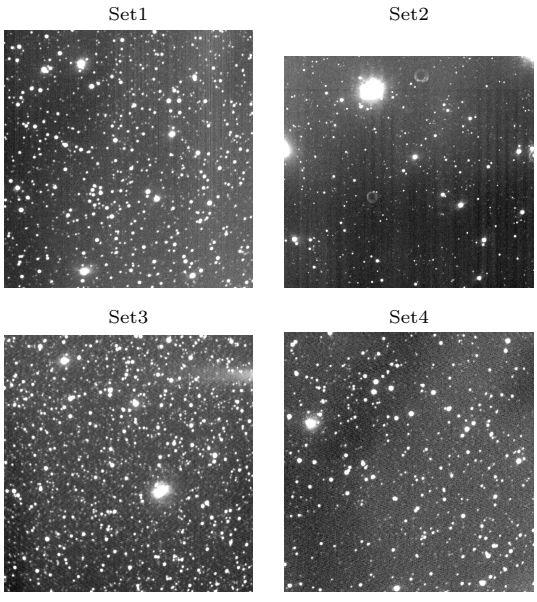

Fig. 12. Co-added images of data sets used for verification of co-adding consistency

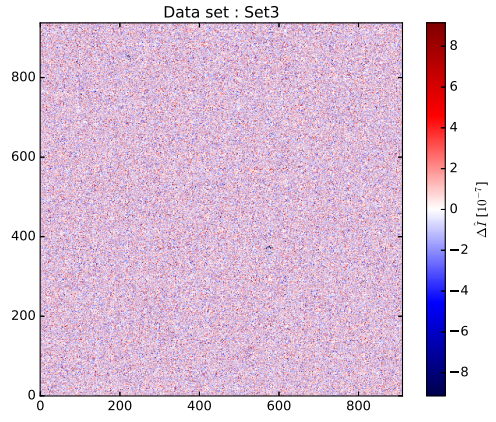

Fig. 13. An image in which each pixel is appropriately colored with $\Delta\hat{I}$. Data set is Set3. No particular pattern appears.

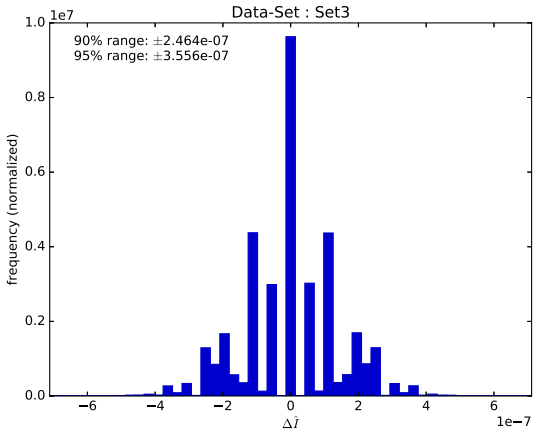

Fig. 14. A histogram of $\Delta\hat{I}$. Data set is Set3. "90(95)% range" is the range in which 90(95)% of Diff of all pixels fall. The value of Diff is discrete, and is about 10^{-7}.

Typical results are shown in figure 13 and figure 14. Figure 13 is a image in which each pixel is appropriately colored with $\Delta\hat{I}$, and figure 14 is a histogram of $\Delta\hat{I}$. In figure 13, no particular pattern is seen. As can be seen from figure 14, $\Delta\hat{I}$ is discrete, and is about 10^{-7}. 32-bit floating point conforming to IEEE754 is assigned 1 bit for a sign, 8 bits for an exponent, and 23 bits for a significand. In the case of a non-zero number, the maximum digit of the significand is always 1. So, the significand is regarded as having 24 bits by assuming that there is always 1 in the 24th digit (IEEE 2008). Therefore, the maximum number of decimal digits that can be represented by 32-bit floating point is $\log_{10} 2^{24} \approx 7$. Since the last digit can easily fluctuate depending on the order of calculations, the maximum number of significant digits is about 6, and this is comparable to our result. Therefore, it can be said that the co-adding function of the new pipeline reproduces the same function as IRAF-`imcombine` within the precision of 32-bit floating point.

3.2 Execution Speed

The procedure for measuring the execution time is described below. Using a set of images that have not been reduced as input, we ran the pipeline 5 times while displaying the execution time with the Unix command `time`. We considered an average of the five output values as the execution time in this condition, and the standard deviation as the error. We define the number of inputs for the three bands as n and the execution time as t.

3.2.1 Comparison with the current pipeline

We measured t while changing n by 30 (10 per band) for the current pipeline, the new one, and the new one running entirely on the CPU without using the GPU. The third is a code in which processing performed by CuPy is rewritten with NumPy to separate the effect of performing processing on the GPU from nonusage of IRAF/PyRAF. Thereafter, we refer to the current pipeline as "Current", the new pipeline as "New(GPU)", and the CPU version of the new pipeline as "New(CPU)". In order to evaluate the difference of the calculation time between the CPU and GPU as purely as possible, we also measured the breakdown of t for New(CPU) and New(GPU). This was performed by acquiring the UNIX time in each processing step using the Python module `time`[16].

We used a set of images acquired with MITSuME-Akeno at August 22, 2018. 188 images per band were captured. In order to make a nice round number, the first input was 180 images per band (i.e. $n = 540$).

Figure 15 is a plot of n and t for each pipeline. It shows that t increases linearly with respect to n for $n \leq 540$. This is because both the file I/O amount and the calculation amount are approximately proportional to the data volume. Figure 16 is a plot of n and t-ratio for each pipeline. The ratio differs depending on n because t has a constant component for n (e.g. starting up the interpreter). When $n = 540$, the ratio of t between Current and New(GPU) is 42. That is, New(GPU) runs 42 times faster. Table 6 shows the breakdown of t for New(CPU) and New(GPU). Since FITS I/O and WCS operation do not use the GPU for calculation even in New(GPU), the results are comparable between New(GPU) and New(CPU). On the other hand, the other processing clearly shows differences among the calculation speed of GPU and CPU, New(GPU) is about 10-1000 times faster than New(CPU). In particu-

[16]Python build-in module that provides time-related functions

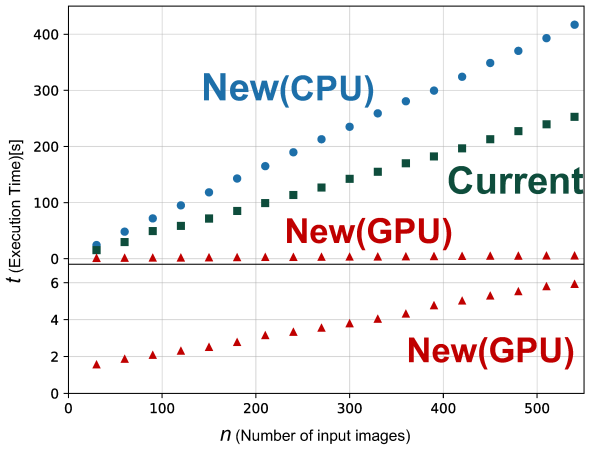

Fig. 15. Plot of n and t for each pipeline. Errors are not drawn because they are comparable to the marker size. It is hard to see the plot of New(GPU) in the above figure, so the scale changed figure is shown below. For $n \leq 540$, t of each pipeline increases almost linearly with n, and New(GPU) is significantly faster than Current and New(CPU). New(CPU) is slower than Current because our implementation of the bicubic-spline interpolation for image alignment is less efficient than IRAF-`imshift`.

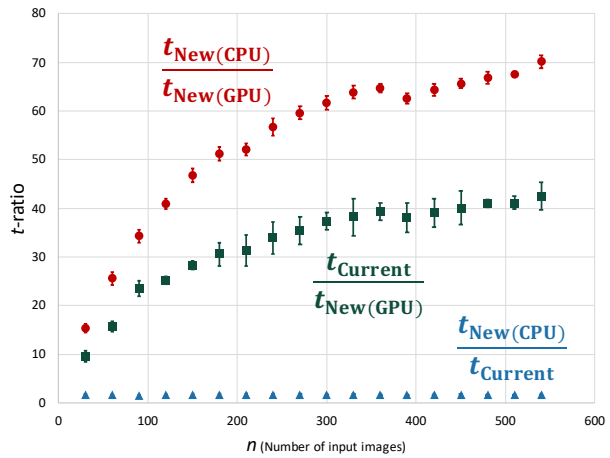

Fig. 16. Plot of n and the ratio of t. When $n = 540$, the ratio of t between Current and New (GPU) is 42. That is, New(GPU) runs 42-times faster.

Table 6. Breakdown of t

$(n = 540)$	New(GPU) [s]	New(CPU) [s]	Ratio[1]
FITS I/O[2]	3.4	2.7	0.81
WCS operation	0.96	0.94	0.97
Bias,Dark,Flat	1.0×10^{-3}	0.74	720
Bad-pixel mask	0.36	10.55	29
Alignment	0.48	4.0×10^{2}	830
Combining	0.14	12	91

[1] New(CPU)/New(GPU)
[2] Including RAM-VRAM transfer time
* Interpreter startup and module initialization time are not shown

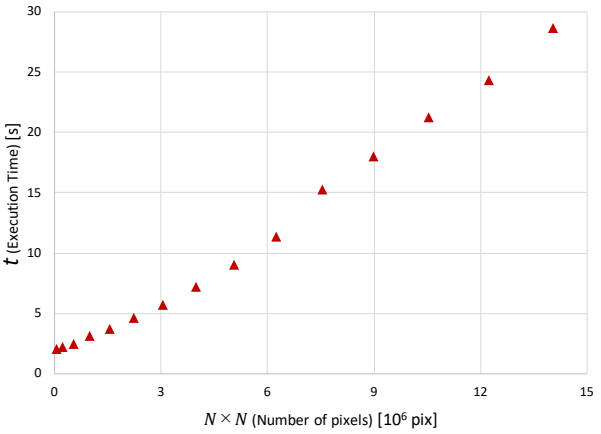

Fig. 17. Plot of $N \times N$ and t. t has a downward convex change with respect to N^2. It seems that linear fitting can be performed up to $N^2 \leq 3 \times 10^6$, but the same fitting cannot be applied to all regions.

lar, since the calculational cost of bicubic-spline interpolation is very large, New(CPU) spends a very long time on alignment. Therefore, if a less costly algorithm such as nearest-neighbor or bilinear interpolation is used, t is expected to be significantly reduced. In our environment, NumPy uses only a single CPU core. However, depending on how NumPy is built, multiple cores can be used, and it is expected to run faster. We note that New(CPU) is slower than Current by factor of 2. This is probably caused by the different implementations of their bicubic-spline interpolations in the image alignment. In the bicubic-spline interpolation, there is a process of solving simultaneous linear equations for each column or row of the image. The number of unknowns of each simultaneous equations is about the same as N, the number of pixels in the each column or row (e.g. $N = 1020$ for MITSuME data), and those simultaneous equations can be expressed as a tridiagonal matrix. The cost of the solution optimized for the tridiagonal matrix is $O(N^2)$, and IRAF-`imshift` probably uses such a solution. The new pipeline solves those equations using the inverse matrix which results in the cost of $O(N^3)$.

3.2.2 Application to various data

We also evaluated the execution time for the various image sizes, assuming the application of this pipeline to observational data other than MITSuME. We generated a set of dummy images in which random values were assigned to $N \times N$ pixels, and measured the execution time t while changing N as long as the VRAM is sufficient. The number of images was fixed at 100 per band (i.e. $n = 300$). The result is shown in figure 17.

Assuming that both the file-reading time and the cal-

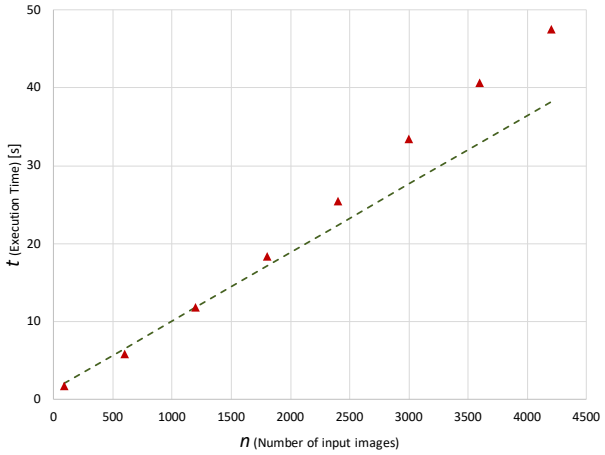

Fig. 18. Plot of n and t. This is the same condition of New(GPU) in figure 15, except that the input is dummy data. The dashed line is the model obtained by linear-fitting the plot of New(GPU) in figure 15. At $n > 1000$, the plot deviates significantly from the line and has a convex downward change.

culation time are basically proportional to the data volume, t is considered to change linearly with respect to N^2. However, in figure 17, t has a convex downward change with respect to N^2. The first conceivable cause is the calculation time of bicubic-spline interpolation. As mentioned in section 3.2.1, our implementation of the bicubic-spline interpolation has a calculational cost of $O(N^3)$. Therefore, this effect does not depend on hardware. Figure 18 shows the result of performing the same measurement while fixing $N = 1020$ and changing n as long as the VRAM is sufficient. This is the same condition as the measurement performed in section 3.2.1, except that the input data is dummy data. When n is large, t has a downward convex change respect to n. The details have not been clarified yet, but we believe that a non-linear effect about the data volume is affecting the execution time. What we currently suspect is the time required for the memory allocation and releasing. However, we could not reproduce that effect by simply allocating or releasing the memory.

4 Summary

To speed up the image reduction processing of MITSuME data, and to establish set of a low-cost, high-speed image-processing methods, we developed a GPU-accelerated image-reduction pipeline. This pipeline uses CuPy, a Python package for GPGPU, to perform image processing on the GPU. Because the library does not provide a corresponding function, there is some processing which was coded by ourselves. However, this pipeline reproduced the functions of the current pipeline with a relative difference of $\lesssim 10^{-5}$. Furthermore, the new pipeline is 42 times faster than the current pipeline when processing 540 images.

Currently, we are performing real-time image reduction of MITSuME observational data with GPU-accelerated pipeline, and using it to check results of prompt follow-up observation of GRBs. At present, the reduced images of this pipeline have been used in 6 reports to GRB Coordinates Network Circular Service (Toma et.al. 2019; Adachi et.al. 2019; Niwano et.al. 2019; Adachi et.al. 2020; Oeda et.al. 2020; Hosokawa et.al. 2020).

We have developed a Python package based on our new pipeline and released it on GitHub[17]. Thus, GPU-accelerated image reduction can be performed with on data from other observatories.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers JP16K13783, JP17H06362 and JP20K04011. This work was also supported by Optical and Near-Infrared Astronomy Inter-University Cooperation Program of the MEXT of Japan, and by JSPS and NSF under the JSPS–NSF Partnerships for International Research and Education.

References

IEEE Standard for Floating-Point Arithmetic," in IEEE Std 754-2008 , vol., no., pp.1-70, 29 Aug. 2008

The LIGO Scientific Collaboration, et.al. 2015, Class. Quantum Grav.32 07400

F. Acernese, M. Agathos, K. Agatsuma, et.al. 2014, arXiv:1408.3978

T. Akutsu, M. Ando, K. Arai, et.al. 2019, Nature Astronomy, 3, pages35–40

K. C. Chambers, E. A. Magnier. N. Metcalfe, et.al. 2016, arXiv:1612.05560

E. A. Magnier, K. C. Chambers, H. A. Flewelling, et.al. 2016, arXiv:1612.05240

S. Miyazaki, Y. Komiyama, S. Kawanomoto, et.al. 2018, PASJ, 70, S1

H. Furusawa, M. Koike, T. Takata, et.al. 2018, PASJ, 70, S3

Eric C. Bellm, Shrinivas R. Kulkarni, Matthew J. Graham, et.al. 2019, PASP, 131, 018002

Frank J. Masci, Russ R. Laher, Ben Rusholme, et.al. 2019, PASP, 131, 018003

S. Sako; R. Ohsawa, H. Takahashi, et.al. 2018, Proc. SPIE, Volume 10702, id. 107020J 17 pp.

Tody Doug, 1986, in Crawford D. L., ed., Instrumentation in Astronomy VI,Vol.627 of Proc. SPIE, The IRAF Data Reduction and Analysis System.p.733

Tody Doug, 1993, Astronomical Data Analysis Software and Systems II, ASP Conf. Ser.Series, Vol.52, 1993, R. J. Hanisch, R. J. V. Brissenden, and Jeannette Barnes, eds., p.173.

T, Kotani, N. Kawai, K. Yanagisawa, et.al. 2005, Il Nuovo

[17] https://github.com/MNiwano/Eclaire

Cimento C, vol.28, Issue 4, p.755
Y. Yatsu, N. Kawai, T. Shimokawabe, et.al. 2007, Physica E, vol.40, Issue 2, p.434
T. Shimokawabe, N. Kawai, T. Kotani, et.al. 2007, GAMMA-RAY BURSTS 2007: Proceedings of the Santa Fe Conference. AIP Conference Proceedings, vol.1000, pp.543
P. Tobias, V. Peter, P. Wolfgang, S. Johannes J, 2009, Journal of Computational Physics, Volume 228, Issue 12, p. 4468-4477.
D. Tuccillo, M. Huertas-Company, E. Decenciere, et.al. 2018, Monthly Notices of the Royal Astronomical Society, Volume 475, Issue 1, p.894-909
M. Tazzari, F. Beaujean and Leonardo Testi, 2018, Monthly Notices of the Royal Astronomical Society, Volume 476, Issue 4, p.4527-4542
Travis E. Oliphant, 2006, USA: Trelgol Publishing
Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux, 2011, Computing in Science & Engineering, 13, 22
Astropy Collaboration, Robitaille T. P, Tollerud E. J, et.al. 2013, Astronomy & Astrophysics, Volume 558, id.A33, 9 pp.
Astropy Collaboration, Price-Whelan. A. M, Sipőcz. B. M, et.al. 2018, The Astronomical Journal, Volume 156, Issue 3, article id. 123, 19 pp.
S. Toma, M. Niwano, R. Adachi, et.al. 2019, GRB Coordinates Network, Circular Service, No.26019
R. Adachi, K. L. Murata, M. Oeda, et.al. 2019, GRB Coordinates Network, Circular Service, No.26597
M. Niwano, K. L. Murata, M. Oeda, et.al. 2019, GRB Coordinates Network, Circular Service, No.26776
R. Adachi, K. L. Murata, M. Niwano, et.al. 2020, GRB Coordinates Network, Circular Service, No.26864
M. Oeda, R. Adachi, K. L. Murata, et.al. 2020, GRB Coordinates Network, Circular Service, No.27236
R. Hosokawa,R. Adachi, K. L. Murata, et.al. 2020, GRB Coordinates Network, Circular Service, No.27429