Invited Paper

High-speed image reconstruction for super-resolution structured illumination microscopy using facile optimization and conversion of reconstruction code in the GPU environment

Kwangsung Oh, [1, *]; Piero R. Bianco [2, *]
[1] University of Nebraska Omaha (United States)
[2] University of Nebraska Medical Center. (United States)
= corresponding authors

ABSTRACT

Super-resolution, structured illumination microscopy (SIM) is an ideal modality for imaging live cells due to its relatively high speed and low photon-induced damage to the cells. SIM consists of two generic components: (i) sample illumination by a sinusoidal pattern and (ii) computational reconstruction of a super-resolution image. The rate-limiting step in observing a super-resolution image in SIM is the reconstruction speed of the algorithm required to form a single image from as many as nine raw images. These reconstruction algorithms impose a significant computing burden due to a complex workflow and a large number of calculations requiring 10-300 seconds per image nullifying real-time imaging. In this mini-review, we show how the approaches we developed to improve Hessian-SIM algorithm reconstruction speed can be used to improve other SIM image reconstruction algorithms. These approaches, which included code improvement, conversion to the GPU environment, and use of cost-effective high-performance computers produce up to 500-fold increases in image reconstruction speed.

Keywords: GPU; CPU; super-resolution imaging; structured illumination microscopy; image reconstruction

1. INTRODUCTION

Super-resolution, structured illumination microscopy (SIM) is an ideal modality for imaging live cells due to its relatively high speed and low photon-induced damage to the cells in comparison to other super-resolution fluorescence microscopy techniques [2, 3]. Structured illumination microscopy and its variants thereof are based on the original wide-field design of Gustafsson [4]. SIM consists of two generic components: (i) sample illumination by a sinusoidal pattern and (ii), computational reconstruction of a super-resolution image [5]. Over the years, intensive research has focused on improving the hardware, the means of sample illumination, the algorithms to reconstruct images, and approaches to increase algorithm reconstruction speed [6-10]. The overarching goal of these combined efforts is to produce an imaging modality that produces super-resolution images in real-time with minimal artifacts [3, 11-14].

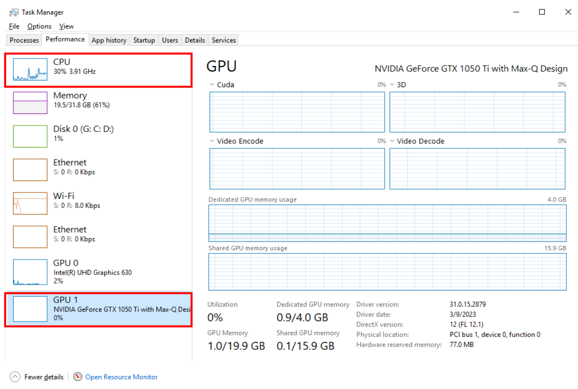
Often the rate-limiting step in observing a super-resolution image in SIM is the reconstruction speed of the algorithm required to form a single image from as many as nine raw images [15, 16]. The speed of execution can be limited by either the code of the algorithms themselves or the computer hardware. Most widely used approaches perform a Fourier transform of the captured images, then perform calculations in Fourier space and once this is done, an inverse Fourier transform is done to produce the super-resolution image. These reconstruction algorithms impose a significant computing burden due to a complex workflow and a large number of calculations to produce the final image [17, 18]. This requires several seconds (10-300 per image) which essentially nullifies real-time imaging [8, 19]. In addition, image reconstruction calculations must be performed with great care as artifacts can be introduced into the final images and this is further complicated by the motion of the cell or organelles during imaging [12, 17, 18, 20-22]. Finally, careful selection of hardware components must be done as these too can be rate limiting in algorithm execution.

Until recently, most image reconstruction algorithms were executed on central processing units (CPUs), where instructions are executed serially. In contrast, the execution of instructions within the graphics processing unit (GPU) environment is done in a massively parallel fashion and is 10- 100-fold faster [23-25]. Thus, and due to the heavy computing burden, it makes sense to reconstruct super-resolution images in the GPU environment. This was first demonstrated, albeit in a complex fashion using three cameras and multiple computers, by Markwirth et al. [26]. More recently, an improved algorithm that used a simplified workflow called Joint Space and Frequency Reconstruction SIM (JSFR-SIM) was developed [27]. While this algorithm is only 2-fold faster than the widely used Wiener SIM, the conversion of code to the GPU environment resulted in a 77-fold improvement in execution speed. However, the CPU-GPU code conversion is not straightforward, and in addition, the vast majority of SIM image reconstruction code is not written by computer scientists, which implies that there could be performance bottlenecks due to the inefficient code.

To improve code execution speed, we developed a set of simple techniques within the framework of MATLAB using the compute-intensive Hessian SIM as the test code and described how to enhance algorithm processing speed [19, 28]. MATLAB is a popular programming language and computing environment for many microscopy researchers as it offers an easy way to write, test, and run many image processing algorithms without background knowledge in computer science. However, the resulting algorithms can suffer from poor performance due to inefficiently written code. When code is optimized, significant speed increases are seen. Execution speeds are further enhanced using GPU-enabled desktop computers with optimized code for the GPU. These lessons were then used to enhance the execution speed of both JSFR- and JSFR-AR-SIM [1, 27]. The results show that the combination of code improvement, conversion to the GPU environment, and use of a GPU-enabled computer, results in a 4- to 500-fold improvement in algorithm execution speed. Importantly, the resulting image quality is identical to that produced by the original algorithm.

2. RESULTS

The scheme to improve code execution is straightforward and uses tools already present in MATLAB. These steps include first identifying both algorithm and hardware bottlenecks as either one or both can contribute to poor execution performance. Details are provided elsewhere but are summarized in the following paragraphs [28]. To identify hardware bottlenecks, the most straightforward approach is to use the Task Manager in Windows and visualize the use of different components during algorithm execution. In the test example shown, that is, using the Hessian SIM algorithm and our baseline computer, the CPU is being slightly used while the GPU is not utilized at all (Fig. 1). These results are directly attributed to how the algorithm was written and is independent of the type of computer used. That is, the code is written inefficiently for the GPU to be idle (less utilized) and designed to be executed on the CPU only, and identical results are observed on more powerful GPU-enabled computers (data not shown; see [28]).



CPU
30% 3.91 GHz

Memory
19.5/31.8 GB (61%)

Disk 0 (G: C: D:)
1%

Ethernet
S: 0 R: 0 Kbps

Wi-Fi
S: 0 R: 8.0 Kbps

Ethernet
S: 0 R: 0 Kbps

GPU 0
Intel(R) UHD Graphics 630
2%

GPU 1
NVIDIA GeForce GTX 1050 Ti with Max-Q Desi
0%

GPU
NVIDIA GeForce GTX 1050 Ti with Max-Q Design

Cuda 0% 3D 0%
Video Encode 0% Video Decode 0%
Dedicated GPU memory usage 4.0 GB
Shared GPU memory usage 15.9 GB

Utilization 0%
Dedicated GPU memory 0.9/4.0 GB
GPU Memory 1.0/19.9 GB
Shared GPU memory 0.1/15.9 GB

Driver version:	31.0.15.2879
Driver date:	3/9/2023
DirectX version:	12 (FL 12.1)
Physical location:	PCI bus 1, device 0, function 0
Hardware reserved memory:	77.0 MB

Figure 1. **Hardware utilization monitoring is easily performed using Task Manager.** A screen capture was performed during the execution of the Hessian-SIM denoising algorithm. Red boxes highlight the CPU and GPU utilization during algorithm execution.

Then, using the following functions in MATLAB, one can identify the code bottlenecks - MATLAB Profiler combined with the *tic* and *toc* functions to determine function execution times. Then the microscopist must determine how frequently memory is accessed and reduce this to a minimum. This follows because memory access latency (> 60 ns) is much higher than either the CPU or GPU (< 1 ns). That is, the code requiring frequent memory access is one of the major performance bottlenecks. Once these issues have been addressed, the code must be carefully examined to determine if it has been inefficiently written. One typical error is conducting redundant operations in a loop that is sometimes repeated hundreds of times. Since such redundant operations waste CPU cycles without any progress, they must be removed from the loops.

In addition to other issues, most software is written to use only a single core within the CPU. Modern CPUs have multiple CPU cores and each can process different tasks independently, i.e., multitasking. Thus, multiple CPU cores can be used for image processing algorithms to improve performance by processing different and independent tasks in parallel, i.e., concurrency. To achieve this, MATLAB provides the addon tool called Parallel Computing Toolbox which allows researchers to exploit multiple CPU cores.

Once code has been optimized to execute as rapidly and efficiently as possible on the CPU, it can be converted to run in the GPU environment. The MATLAB Parallel Computing Toolbox also allows researchers to easily exploit the GPUs. As for the CPU code, the GPU code must again be optimized to fully exploit GPU performance via massively parallelism.

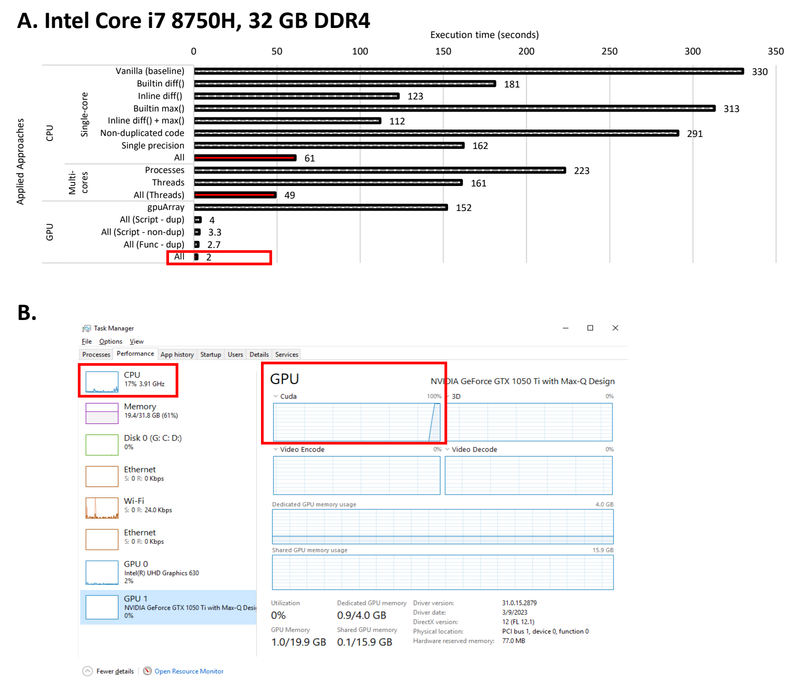


Figure 2. Impact of improved code on algorithm execution performance. (A), Analysis of individual steps and (B), The Windows Task manager shows the impact of improved code on CPU and GPU utilization.

For example, image data need to be stored in CPU memory and GPU memory to be processed and transferred between them. However, memory access is slow operation which makes CPU and GPU wait and the transfer between CPU and GPU memory is rate-limiting due to the limited bandwidth between these components. Thus, frequent memory access and data transfer between the CPU and the GPU during algorithm execution incurs performance overhead and thus must be avoided. To avoid unnecessary memory access and data transfer, all data required for image processing algorithms in CPU memory can be copied into GPU memory a priori, i.e., pre-allocating. This makes all data processing done in the GPU without additional CPU memory access, which improves performance significantly.

The outcome of the improved code on algorithm execution speed is shown in Figure 2. For performance comparison, we use a single 128 x 256 x 180 image stack. At the top of panel A, the improved code for the CPU executes 5-fold faster than the vanilla code (baseline) on a single CPU core. When multiple CPU cores are used, code executes 7-fold faster compared to unmodified code. Finally, when the code is optimized for the GPU environment, the algorithm that took 330 secs to execute is now accomplished in 2 secs, a 165-fold improvement. In addition, the CPU is now used at only 17% of capacity (down almost 2-fold; Fig. 1) and the GPU which was not used at all, is now working at full capacity (Fig. 2B), which shows that the GPU has now become a performance bottleneck.

Additional improvements are observed when powerful GPU-enabled computers are used (Table 1) [28]. Here two machines were built. The first used Intel I9 technology and had a GPU, while the second used an AMD Ryzen Threadripper CPU and had two GPUs. For the Intel-based machine, the algorithm takes 670 msec to execute while the AMD- based machine requires 800 msec. We expect further improvements in execution speed for the AMD machine as we did not take full advantage of the two GPUs due to the limited support of MATLAB for multiple GPUs (data not shown). Collectively the combination of improved code, conversion to the GPU environment, and use of powerful GPU-enabled computers results in a 490-fold increase in algorithm execution speed.

Table 1.
The combination of GPU-optimized code and improved hardware produces maximum performance increases in algorithm execution speed.

Algorithm executed	Baseline computer[a]	Intel I-9 computer	AMD Ryzen Computer	Baseline PC (%)		I-9 PC (%)		Ryzen PC (%)	
				CPU	GPU	CPU	GPU	CPU	GPU
Initial algorithm speed (sec)	330	133	175	30	0	18	1	9	1
Improved; CPU (sec)	49	19.5	25	ND[b]		ND		ND	
Improved; GPU (sec)	2	0.67	0.8	17	100	7	82	ND	

a. The three computers used for testing are: (1), Baseline - Dell (XPS 15 9570); Intel Core i7-8750H, 32GB of DDR 4 RAM; 1 SSD (2TB Samsung SSD 970 EVO Plus one NVIDIA GeForce GTX 1050 Ti with Max-Q Design. The operating system is Windows 10. (2), Dell (Precision 3660) with an Intel W680 (Alder Lake-S PCH) motherboard; an Intel Core i9-12900K CPU, 64GB of DDR5 RAM; 2 SSDs (1TB NVMe SK Hynix and 4TB Seagate ST4000DX005) and one NVIDIA RTX 3090 graphics card with 24 GB of GPU memory. The operating system is Windows 11. (3), DigitalStorm computer with an ASUS ROG Zenith II Extreme Alpha motherboard; an AMD Ryzen Threadripper 3990X CPU, 128GB of DDR4 RAM; 3 SSDs (1TB Samsung 970 EVO Plus; 2TB Samsung 860 Pro and a 4TB Samsung 860 Pro) and two, NVIDIA RTX A6000 graphics cards with 48GB of GPU memory each. The operating system is Windows 11.

b. ND, not done.

Due to time constraints, only some of the above-mentioned improvements could be applied to the JSFR- and JSFR-AR-SIM code (Table 2) [1, 27]. Even with these limited improvements, these data show that each algorithm is executed 20- to 60-fold faster in the GPU environment as compared to the CPU. These speed improvements mean that the combination of acquisition and image processing produces a super-resolution image in 67 to 88 msec. Consequently, using these two modalities, all microscopy is done in super-resolution imaging mode only. In contrast, previous SIM implementations required that an initial field of view be located in widefield mode and then switch to SIM for super-resolution imaging. This is a laborious and time-consuming process that is now eliminated.

Table 2.

The impact of improved code and implementation of the GPU environment on algorithm execution speed.

Input image size	Acquisition time (ms)	Reconstruction time of JSFR-SIM (ms)		Reconstruction time of JSFR-AR-SIM (ms)	
		CPU	GPU	CPU	GPU
1024×1024	45.0	1401.9±15.0	43.3±0.8	1340.9±70.8	21.5±6.1
512×512	22.5	293.6±4.7	10.2±0.7	274.9±7.3	5.7±4.5
256×256	11.3	73.0±0.7	4.5±0.2	68.1±1.7	3.1±0.1

Once algorithm execution speed has been improved the resulting image quality must be assessed to determine if it is unchanged compared to the original. To do this, images can be compared byte by byte and separately, using Image J (Fig. 3) [28]. This analysis shows that the image quality is identical and the only difference is the execution speed of the image reconstruction algorithm

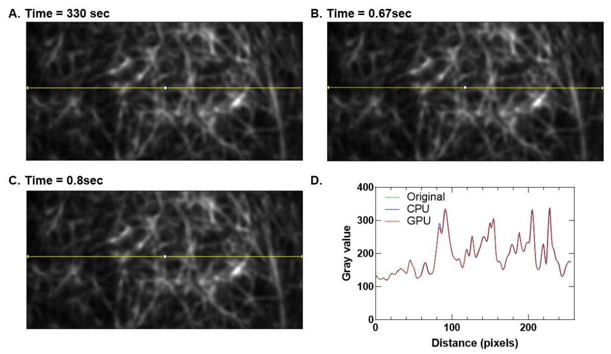


Figure 3. Image analysis reveals that the increase in algorithm execution speed does not come at the expense of image quality. A-C, Representative image frames produced using the Hessian SIM algorithm. A, Initial algorithm. B, GPU-enhanced algorithm executed on the Intel-based machine. C, The same algorithm executed on the AMD-based computer. D, line profile analysis of the yellow lines in A-C.

To further demonstrate this, a comparison of images obtained using the GPU-enhanced code is presented [1](Fig. 4). These images are within experimental error, identical but are reconstructed at significantly greater speeds relative to the unenhanced algorithm. Compare Fig 4B and D to Fig. 4C which were reconstructed using GPU-enhanced JSFR- and JSFR-AR-SIM and CPU-executed HiFi-SIM, respectively.

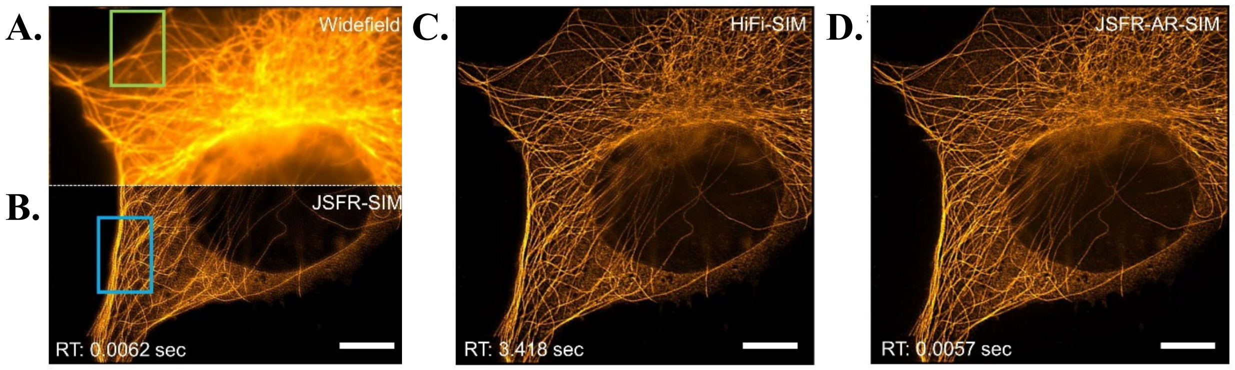


Figure 4. Improved algorithms executed in the GPU environment produce identical images but at significantly more rapid rates. A-D, Images of microtubules stained with Cy3B. Algorithm execution speed is shown at the bottom left of each image. The scales bars are 5µm [1].

3. CONCLUSIONS

The primary conclusions of this work are that improved code implemented on GPU-enhanced computers results in significantly faster algorithm execution. For structured illumination microscopy, the reconstruction of super-resolution images is sufficiently rapid to enable the microscopist to image in only super-resolution mode, simplifying the workflow while simultaneously obtaining images in less than 90 msec. Due to time constraints, only a limited number of improvements in algorithms could be implemented and the GPU-enhanced computers could not be used for the super-resolution imaging. Consequently, we anticipate further speed improvements in algorithm execution speed once all changes are implemented and the enhanced computing environment is taken advantage of.

4. REFERENCES

1. Wang, Z., et al., *Rapid, artifact-reduced, image reconstruction for super-resolution structured illumination microscopy.* Innovation (Camb), 2023. **4**(3): p. 100425.
2. Hirano, Y., A. Matsuda, and Y. Hiraoka, *Recent advancements in structured-illumination microscopy toward live-cell imaging.* Microscopy (Oxf), 2015. **64**(4): p. 237-49.
3. Heintzmann, R. and T. Huser, *Super-Resolution Structured Illumination Microscopy.* Chem Rev, 2017. **117**(23): p. 13890-13908.
4. Gustafsson, M.G., *Surpassing the lateral resolution limit by a factor of two using structured illumination microscopy.* J Microsc, 2000. **198**(Pt 2): p. 82-7.
5. Kner, P., et al., *Super-resolution video microscopy of live cells by structured illumination.* Nat Methods, 2009. **6**(5): p. 339-42.
6. Wu, Y. and H. Shroff, *Faster, sharper, and deeper: structured illumination microscopy for biological imaging.* Nat Methods, 2018. **15**(12): p. 1011-1019.
7. Zhao, T., et al., *Advances in High-Speed Structured Illumination Microscopy.* Frontiers in Physics, 2021. **9**.
8. Muller, M., et al., *Open-source image reconstruction of super-resolution structured illumination microscopy data in ImageJ.* Nat Commun, 2016. **7**: p. 10980.
9. Ma, Y., Wen, K., Liu, M., Zheng, J., Chu, K., Smith, Z. J., Liu, L., Gao, P., *Recent advances in structured illumination microscopy.* Jphys Photonics, 2021. **3**: p. 024009.
10. Curd, A., et al., *Construction of an instant structured illumination microscope.* Methods, 2015. **88**: p. 37-47.
11. Smith, C.S., et al., *Structured illumination microscopy with noise-controlled image reconstructions.* Nat Methods, 2021. **18**(7): p. 821-828.
12. Fan, J., et al., *A protocol for structured illumination microscopy with minimal reconstruction artifacts.* Biophysics Reports, 2019. **5**(2): p. 80-90.
13. Pospíšil, J., K. Fliegel, and M. Klíma, *Analysis of image reconstruction artifacts in structured illumination microscopy.* Proc. SPIE 10396, Applications of Digital Image Processing XL, 1039632 2017. **10396** p. 1-12.
14. Wen, G., et al., *High-fidelity structured illumination microscopy by point-spread-function engineering.* Light Sci Appl, 2021. **10**(1): p. 70.
15. Sahl, S.J., et al., *Comment on "Extended-resolution structured illumination imaging of endocytic and cytoskeletal dynamics".* Science, 2016. **352**(6285): p. 527.
16. Heintzmann, R. and M.G. Gustafsson, *Subdiffraction resolution in continuous samples.* Nature Photonics, 2009. **3**(7): p. 362-364.
17. Wicker, K., *Non-iterative determination of pattern phase in structured illumination microscopy using auto-correlations in Fourier space.* Opt Express, 2013. **21**(21): p. 24692-701.
18. Chu, K., et al., *Image reconstruction for structured-illumination microscopy with low signal level.* Opt Express, 2014. **22**(7): p. 8687-702.
19. Huang, X., et al., *Fast, long-term, super-resolution imaging with Hessian structured illumination microscopy.* Nat Biotechnol, 2018. **36**(5): p. 451-459.
20. Schaefer, L.H., D. Schuster, and J. Schaffer, *Structured illumination microscopy: artefact analysis and reduction utilizing a parameter optimization approach.* J Microsc, 2004. **216**(Pt 2): p. 165-74.
21. Forster, R., et al., *Motion artefact detection in structured illumination microscopy for live cell imaging.* Opt Express, 2016. **24**(19): p. 22121-34.
22. Zhou, X., et al., *Image recombination transform algorithm for superresolution structured illumination microscopy.* J Biomed Opt, 2016. **21**(9): p. 96009.
23. Gong, H., W. Guo, and M.A.A. Neil, *GPU-accelerated real-time reconstruction in Python of three-dimensional datasets from structured illumination microscopy with hexagonal patterns.* Philos Trans A Math Phys Eng Sci, 2021. **379**(2199): p. 20200162.
24. Lu, G., et al., *A real-time GPU-accelerated parallelized image processor for large-scale multiplexed fluorescence microscopy data.* Front Immunol, 2022. **13**: p. 981825.
25. Aydin, M., Uysalli, Y., Ozgonul, E., Morova, B., Kiraz, A., *An LED-based Super Resolution GPU Implemented Structured Illumination Microscope*, in *Single Molecule Spectroscopy and Superresolution Imaging XIII*, I. Gregor, Koberling, F., Erdmann, R., Editor. 2020, SPIE: San francisco, CA. p. 1124610.

26. Markwirth, A., et al., *Video-rate multi-color structured illumination microscopy with simultaneous real-time reconstruction.* Nature communications, 2019. **10**(1): p. 4315.
27. Zhaojun Wang, T.Z., Huiwen Hao, Yanan Cai, Kun Feng, Xue Yun, Yansheng Liang, Shaowei Wang, Yujie Sun, Piero R. Bianco, Kwangsung Oh, Ming Lei *High-speed image reconstruction for optically sectioned, super-resolution structured illumination microscopy.* Advanced Photonics, 2022. **4**(2): p. 026003 (2022).
28. Oh, K. and P.R. Bianco, *Facile conversion and optimization of structured illumination image reconstruction code into the GPU environment.* International Journal of Biomedical Imaging, 2023. **submitted**.