2022 IEEE High Performance Extreme Computing Conference (HPEC) | 978-1-6654-9786-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/HPEC55821.2022.9926403

A Scalable Inference Pipeline for 3D Axon Tracing Algorithms

Benjamin Fenelon[1], Lars A. Gjesteby[1], Webster Guan[2], Juhyuk Park[2], Kwanghun Chung[2], Laura J. Brattain[1]
[1]Human Health & Performance Systems, MIT Lincoln Laboratory, Lexington, MA, USA
[2]Institute for Medical Engineering and Science, Massachusetts Institute of Technology, Cambridge, MA, USA
{benjamin.fenelon, lars.gjesteby, brattainl}@ll.mit.edu
{wjguan, juhyuk, khchung}@mit.edu

Abstract—**High inference times of machine learning-based axon tracing algorithms pose a significant challenge to the practical analysis and interpretation of large-scale brain imagery. This paper explores a distributed data pipeline that employs a SLURM-based job array to run multiple machine learning algorithm predictions simultaneously. Image volumes were split into N (1-16) equal chunks that are each handled by a unique compute node and stitched back together into a single 3D prediction. Preliminary results comparing the inference speed of 1 versus 16 node job arrays demonstrated a 90.95% decrease in compute time for 32 GB input volume and 88.41% for 4 GB input volume. The general pipeline may serve as a baseline for future improved implementations on larger input volumes which can be tuned to various application domains.**

Index Terms—**Brain mapping, machine learning, axon tracing, high performance computing, parallel processing**

I. Introduction

As advances in high throughput and high resolution brain imaging continue, the prospect of mapping a whole human brain at cellular and sub-cellular resolutions becomes increasingly more achievable. While manual tracing of axons may be considered as the gold standard of accuracy, this laborious task will not scale to reconstructing connections between the tens of billions of neurons throughout the human brain. A major goal in large-scale brain mapping is to develop image processing pipelines that can perform automatic segmentation of neuron structures from raw imagery [1] of massive data sizes ranging from gigabytes to petabytes. There is a critical need to optimize automated neuron segmentation methods that rival human performance. Machine learning (ML) algorithms have shown promise in automated analysis, but the current long ML training and inference times for large volumes create a bottleneck to the downstream interactive analysis.

Recent work has demonstrated methods toward high-throughput imaging and algorithm pipelines for various tasks in brain mapping. Most of the existing work focuses on improving ML model training performance. Very few address the long inference time, which is still a significant gap. One study looked at segmentation of unmyelinated axon fibers of vagus and pelvic nerves in rats imaged with transmission electron microscopy [2]. The image slices were very thin, between 70-90 nm, and each image was divided into a number of tiles ranging from 20 to 200. For a single slice, the inference time was reported as 54-250 seconds, depending on the tile size (256-768 pixels). While this work shows effectiveness for the prescribed task, it does not go as far as processing dense 3D volumes from light sheet microscopy that cover larger fields of view. Another study demonstrated a pipeline for whole brain fluorescence imaging of primate brains combined with long-range axon tracing [3]. The authors reported automated tracing speeds of 10-30 mm/h, with approximately 4 terabytes of data accessed during tracing, which is a promising step in the right direction.

In our paper, a distributed data pipeline is developed for simultaneous inference of a 3D ML-based axon tracing algorithm across multiple lightsheet microscopy image volume chunks. The experiments compare various image volume sizes and numbers of compute nodes to demonstrate substantial speed-up and scalability of voxel processing time.

II. Materials and Methods

A. Dataset Description

The dataset used in this study has been described in recent work. Briefly, a 1-mm-thick human brain tissue-gel was immunostained with anti-NFH (neurofilament heavy chain) antibody after tissue clearing with the mELAST technique [4]–[6]. The tissue was imaged with light sheet microscopy using a 15x objective after 3x physical expansion, and the resulting volume measured 18469 $\times$ 7571 $\times$ 9422 voxels with voxel dimensions 0.299 x 0.422 x 0.299 μm. Input volumes for this paper were extracted from the full dataset by specifying unique index ranges and converting to a stacked image format (.zarr to .tiff) with the following sizes:

- 256 $\times$ 256 $\times$ 256 voxels (64 MB)
- 1024 $\times$ 1024 $\times$ 1024 voxels (4 GB)
- 2048 $\times$ 2048 $\times$ 2048 voxels (32 GB)

Each input volume was preprocessed as described in prior work [7], by clipping the lowest and highest 0.01% values, applying a median filter, scaling the values to [0,1] with min-max normalization, and converting to the model input format (.h5). After preprocessing, each input volume was treated as an independent dataset for ML model prediction, passed as samples of 128 $\times$ 128 $\times$ 64 voxels. Due to this fixed sample size, the 256-cubed voxel input volume could only be split across a maximum of 4 compute nodes.

B. Axon Tracing Algorithm

The deep learning-based axon tracing algorithm utilized in this study was previously developed in [8]. Using a dataset with focus on parvalbumin positive neurons from the globus pallidus externus (PVGPe), the axon centerline detection algorithm achieved improved performance through a topologically aware loss function. The best-performing architecture was a 3D U-Net [9] with a centerline dice (clDice) loss function [10]. This model was selected for inference on the 3 aforementioned input volumes to establish a baseline time and investigate the relationship between performance (time) and image size (voxels).

The inference algorithm accepts preprocessed axon imagery and generates predictions of the same data format using a selected pretrained ML model. A SLURM batch job [11] allocates 1 compute node with 2 GPUs to the model and unpacks the input data. Before loading, the image samples are cropped using a sliding window (50% overlap) and blended using a 3D Hann window to minimize prediction errors along the edge of the receptive field. The model can then process the 8 input batches as parallel microbatches using DataParallel [12]. Additional steps may be taken in post-processing to convert predictions to visualizations, though not evaluated in this study.

C. High performance computing environment

In our high performance computing (HPC) system, we were able to access compute nodes each equipped with 2 Intel Xeon Gold 6248 CPUs (40 total cores) and 2 NVIDIA Volta V100 GPUs accessible through a SLURM workload manager [11]. With exclusive access, a single node provides 384 GB of node RAM and 64 GB of GPU RAM (32 GB each).

A single compute node with 2 Volta GPUs was used for the baseline job submission and extended to a job array for N simultaneous submissions of the inference job. A SLURM job array submission allocates the user requested resources for N jobs with indices $\{0, \ldots, N\text{-}1\}$. In other words, the array is an efficient loop of N independent jobs, each with one of N compute nodes. This is possible from a single array submission because each job is performing the same operation on unique inputs. If there are not enough nodes available to the user, the remaining jobs will be queued until resources are available. The maximum allocation of GPUs for a base user of TX-Green is 32, therefore the experiments of this paper explore a maximum job array size of 16 compute nodes.

D. Data Inference Pipeline

The proposed data processing pipeline is intended to speed up inference time, or the elapsed time from preprocessed data to saved prediction, by splitting the task into three modular components. Due to the size of data, the RAM necessary for processing may exceed the single node capacity and fail to complete the task. By splitting into three separate steps, each output was saved to disk storage before proceeding to the next operation and preserved in case of a downstream failure. These saved checkpoints also enabled user validation after each operation to ensure correctness. If RAM restrictions were removed and validation unnecessary, the pipeline may be implemented as a single job that performs all operations under one master.

Illustrated in Fig.1, the HPC pipeline consists of 3 independent job submissions associated with one raw image input volume executed as sequential operations: split, run, and stitch. The graphic itself includes examples of a raw image (2048-cubed voxels) and a post-processed final prediction. The cubes are oversimplified for visualization purposes depicting 3 squared frames imposed on 3 faces of a cube. The input and output arrows outside of the boundary labeled HPC pipeline represent the preprocessing and postprocessing steps as the preceding and succeeding operations of the proposed pipeline. Therefore, the scope of this paper's experiments are within this green dotted boundary. All scripts within this boundary are stored and executed on our HPC cluster along with the referenced axon centerline algorithm.

1) Split Operation: The preprocessed input and prediction output data have array-like structures [13], which simplify split and stitch to NumPy operations. To split, N adjacent chunks were created using np.array_split [14] along the z-axis. For values of N that could not equally divide the input, 2 array shapes were selected until no data was lost. All chunks were saved as standalone files by a unique filename index that corresponded to the order of the split operation. Here the operation may be reversed by assembling files from index 0 to N-1 until the filesize is equivalent to the unsplit input volume.

2) ML Inference: The inference procedure was a job array that requests N compute nodes, each with 2 Volta GPUs and 40 cores of Xeon Gold CPUs, to handle the chunks of the input volumes. Since each chunk corresponded to an independent node, the environment variable "SLURM_ARRAY_TASK_ID" indicated which filename to process. After performing model inference, a prediction was saved to a file similar to each chunk, including a unique index identifier that preserved split order.

While this approach was a long running batch job, future implementations may seek to make the pipeline many modular steps that can take advantage of node-based job scheduling for short running jobs [15]. Developing a solution with an emphasis on small repetitive tasks, the inference step can be further adapted for smaller chunks, model batch input, and less resources per operation. However, to maintain a comparison with the performance of the previous axon detection work [8], the aforementioned parameters were preserved.

3) Stitch Operation: Once the inference stage was finished, the stitch job was submitted to reconstruct the original shape of the input volumes. The predictions were sorted by the split index in the filename and data was concatenated along the z-axis. The final stitch was saved as an independent file of array-like data with the same shape as the preprocessed input. By caching the output of each step to disk storage as a saved file, the storage requirement would be the sum of input size, split size, inference size, and final result. While necessary in

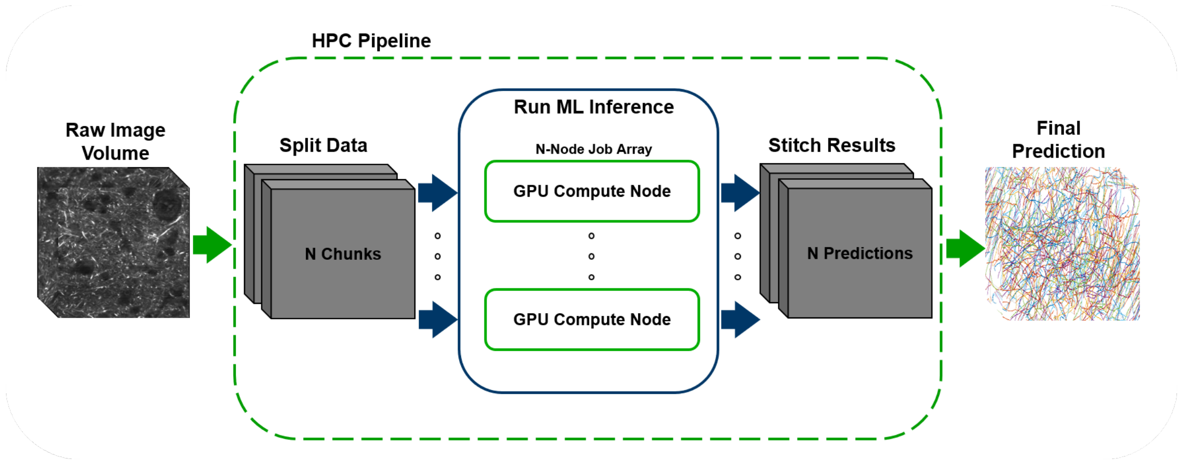

Fig. 1. Overview of proposed distributed and parallel inference pipeline.

testing the pipeline for validity, future implementations may seek to cache the step outputs in RAM or delete after use.

III. Experimental Results

A. Modular Component Evaluation

The proposed image processing pipeline sought to address the long inference times associated with large input volumes by distributing across compute nodes. To understand the relationship between inference time and compute nodes, the selection of N was bounded to the range of 1 to 16. The upper bound of 16 was set by resource constraints and lower bound of 1 set to compare scaled performance. The remaining numbers 2, 4, 6, and 8 were selected to observe trends across magnitudes and uneven splits. With 3 distinct image input volumes and set of N, each step of the pipeline was measured for elapsed time and summed for a total start to end time. Due to the modular structure, the order of job submission varied and certain long running models were not run immediately after split.

Each compute node in the job array has a unique job ID that was monitored with SLURM bash and analyzed for elapsed time after successful completion. The code for each step logged computational runtime, however the elapsed time outputted by *sacct* (SLURM) was more indicative of wall time (the actual time taken from the start to the end) and therefore a better indicator of usability. In the duration of these experiments, the average difference between these times was 2.64s and the maximum was 15s, which was the 4 node split of 1024-cubed voxels. Considering this, there was negligible difference in the times and the longest was selected for analysis.

Since resource allocation can vary, the pipeline job submissions were subject to node availability and limited by the number of ongoing processes. The length of the job arrays contributed to a variability in times as each node was assigned simultaneously, but completed at a different time. Each of these end times were recorded and the longest was selected for sum time, since each step of the pipeline must fully complete before moving to the next. For more suggestive times, each experiment would ideally be performed multiple times and averaged or pruned of outliers.

B. Input Volume Selection

Originally, the single node pipeline included a split and stitch step, however for practical purposes there was no reason to perform these two operations, and thus they were redacted from results. However, the size of the split and stitch files were consistent with the untouched alternatives and confirmed the preservation of data in the pipeline. This continued with the 6 node split, where the shape and size of data were preserved despite an uneven split along the z-axis. The resulting chunks were of 2 different sizes (2048-cubed: 5.32/5.34 GB, 1024-cubed: 680/684 MB). Again, the smallest input volumes, 256-cubed voxels, was processed by a maximum of 4 distributed nodes due to the crop size of the model preventing a higher order of splits. The range of inference input sizes for each input volume after split among 1-16 nodes was as follows:

- $2048 \times 2048 \times 2048$ voxels: 32 GB - 2 GB
- $1024 \times 1024 \times 1024$ voxels: 4 GB - 256 MB
- $256 \times 256 \times 256$ voxels: 64 MB - 16 MB

The largest input volume in this experiment was less than 1% of the total NFH volume and was not characteristic of the entire imagery. However, for scale, the annotated training data in the original implementation was 256-cubed voxels [8], the smallest input volume selected for inference and $512\times$ smaller than the largest. The experiments here can still serve as a useful indication of scalability on much larger volumes.

C. Pipeline Performance

Selected results for experiments across 4 distinct node counts were recorded in Table I. Image dimensions separate the 3 input volumes investigated and each include a baseline (no pipeline), 1 node, 2 node, and best node performance. For a given input volume, the best node refers to the pipeline with shortest time to stitched prediction, or sum of elapsed times for each component of the pipeline. In all cases, the largest

TABLE I
DISTRIBUTED PIPELINE TIME SUMMARY

Image Voxel Size	Node Count	Per Node Input Size	Split Time	Inference Time	Stitch Time	Total Time	Time Per Voxel (μs)
$256 \times 256 \times 256$	Baseline	64 MB	X	00:08:26	X	00:08:26	30.16
	1 Node	64 MB	X	00:08:13	X	00:08:13	30.46
	2 Node	32 MB	00:00:05	00:04:51 (+4.81%)	00:00:08	00:05:04	18.12
	4 Node	16 MB	**00:00:04**	**00:03:00** (+6.11%)	**00:00:03**	**00:03:04**	**10.97**
$1024 \times 1024 \times 1024$	Baseline	4 GB	X	05:43:13	X	05:43:13	19.18
	1 Node	4 GB	X	05:44:35	X	05:44:35	19.26
	2 Node	2 GB	**00:00:06**	02:58:49 (+0.34%)	**00:01:47**	03:00:42	10.10
	16 Node	256 MB	00:00:43	**00:34:11** (+7.22%)	00:04:53	**00:39:47**	**2.22**
$2048 \times 2048 \times 2048$	Baseline	32 GB	X	42:48:17	X	42:48:17	17.88
	1 Node	32 GB	X	42:29:45	X	42:29:45	17.77
	2 Node	16 GB	00:03:40	21:38:14 (+0.11%)	**00:18:31**	22:00:05	9.22
	16 Node	2 GB	**00:01:45**	**03:25:34** (+4.68%)	00:25:07	**03:52:26**	**1.62**

*Bold indicates the best time (HH:MM:SS) per voxel size including unlisted node counts

possible number of nodes for a given input volume yielded the best performance.

The inference times include the percent difference between first and last node completion in a given pipeline. This percentage number steadily increased with N for all input volumes and highlights the variability in distributing across multiple workers. In calculation, the maximum overall difference by value was 00:09:37 (HH:MM:SS) (16 node pipeline on 2048-cubed voxels). The difference may be on the resource level or due to varying image density.

The split, longest inference, and stitch times were summed for each node count to compute the total time. This time demonstrated the overall elapsed time from a preprocessed input volume to final prediction. The total time was also divided by the number of voxels in a given input volume to determine the time to process a single voxel. For each pipeline, the aggregate time, or summation of all elapsed times, was larger than each baseline. Therefore, the expense for a significant decrease in wall time was an increase in total computation time.

1) Split Performance: The splitting operation iterated over the range of 1-16 nodes to save distinct chunks. The per-node input size was the size of each distinct chunk. The split times did not reflect a strict pattern in relation to node count, despite the table suggesting a mostly decreasing relationship. This conclusion was reached when the direction of change (positive or negative) for split time was not shared for any span of node counts across all 3 input volumes. However, split time strictly increased with voxel size for all node counts. By decreasing input volume size, the ranges for split time were 00:01:55, 00:01:00, and 00:00:01. The lack of strict correlation between number of splits and time did not present a clear bottleneck in relation to the tested variables. To uncover unseen trends, future work may be interested in performing the split operation multiple times for each node count.

2) Inference Performance: In this study, there were two bases for comparison. A baseline non-pipeline inference time with no split and stitching operations, and a single node inference with the prediction implementation used in the pipeline. The single node pipeline was a refactored iteration

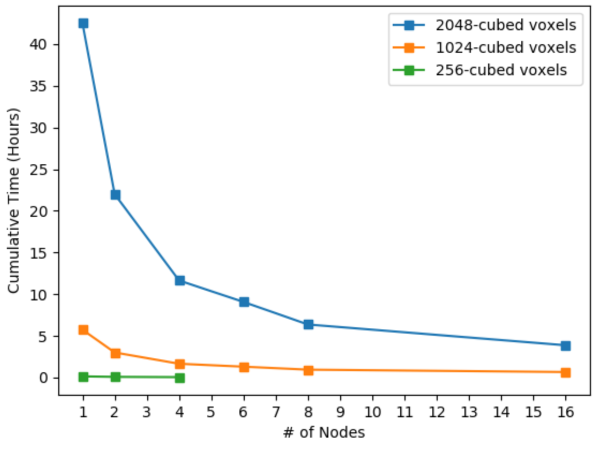

Fig. 2. Total inference time across number of compute nodes in job array.

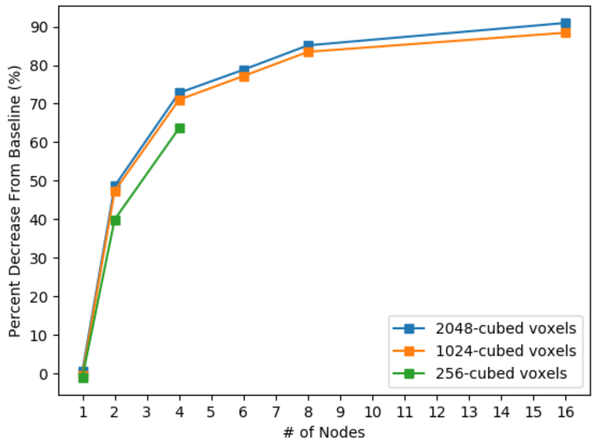

Fig. 3. Percent time decrease from non-pipeline baseline across number of compute nodes.

of the baseline with a time logger function and environment variable checker that updated the input path. The baseline was submitted as a batch job and the single node pipeline submitted as a single node job array. Using both as baseline numbers, all job array submissions were compared to each other as well as to the first recorded time. The trend between nodes and inference times was inverse, but not linear as each recorded inference time had less scaled improvement. The total time was plotted in Fig. 2 as a function of the number of nodes across voxel sizes. The improvement decays across node count and increased with input volume voxel size. If scaled without decay, each 16 node pipeline would perform 16× faster than the 1 node. In experimentation, the improvement factor for inference times on 2, 4, 6, 8, and 16 nodes was as follows, respectively:

- 2048 × 2048 × 2048 voxels: 1.93x, 3.65x, 4.69x, 6.68x, 10.98x
- 1024 × 1024 × 1024 voxels: 1.91x, 3.47x, 4.40x, 6.07x, 8.66x
- 256 × 256 × 256 voxels: 1.68x, 2.78x

The relationship between improvement scale and voxel size further increased across node count. The exploration of higher node counts and larger input volumes may add to these trends and suggest optimal selection of N given image dimensions. The least decay for any node count was with smallest N, which was 2 for all input volume sizes, with the closest being 1.93x improvement for 2048-cubed voxels.

In Fig. 3 each experiment time was compared to the baseline by percent decrease, as a measure of separation from the previously accepted time. The decay mentioned above was illustrated in the plot as the slope between points decreased as the number of nodes increased, but the point values increased with voxel size. Likewise, Fig. 2 preserved this information as each slope between points was less steep over time, though the separation between voxel sizes was greater due to the increased magnitude of total time. The overall decrease in time to prediction suggested that when executing multiple job arrays simultaneously, multiple tasks will be completed sooner and free up the resources for more jobs/users.

3) Stitch Performance: The stitch process was an axis-wise concatenation of the sorted predictions. Similar to the split time, there was high variability in stitching. The stitch time of the 2048-cubed and 1024-cubed voxel input volumes increased from 2 nodes to 16 nodes, with a difference of 00:06:36 and 00:03:06, respectively. In contrast, the stitch time of the 256-cubed voxel input volume decreased from 2 nodes to 4 nodes by 5 seconds. The maximum stitch times for each dataset, from largest to smallest input volume, were for 6 nodes, 4 nodes, and 2 nodes, respectively. The magnitude of split time increased with voxel size across all nodes. The magnitude of change between average stitch time was 36.18× from 256-cubed to 1024-cubed voxels and 6.90× from 1024-cubed to 2048-cubed voxels. Stitch time for a given node count was unpredictable given the current tested variables, but increased with input volume size.

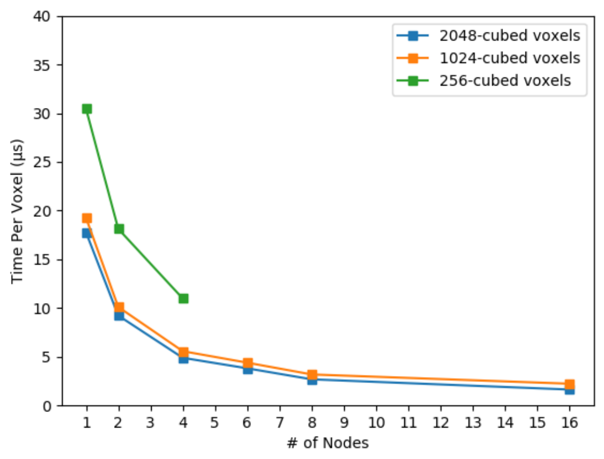

Fig. 4. Time per voxel for each input volume across number of compute nodes.

4) Total Time for Pipeline: The most influential term in the summation of total time was inference time due to its proportion. As described above, no clear linear trend exists for split and stitch times across node count. For this set of input volumes and nodes, the best inference time led to the best total time. Yet given enough nodes and the continuation of this trend, the total time for a input volume may be limited by the stitch operation, the second most time-consuming step of the pipeline and more than 5× larger than splitting. This implied that future improvements to inference time will decrease overall time and increase the magnitude of improvement over the baseline.

Depicted in Fig. 4, the time per voxel (TPV) highlights the scalability of the pipeline in processing data of different sizes. It can be observed that there is a sharp decrease in TPV as the number of nodes increases. In addition, the TPVs of 1024 cube and 2048 cube track each other nicely without significant changes. This indicates the potential of maintaining consistent TPV as the input volume size increases.

IV. Discussion and Future Work

This paper demonstrated a scalable ML model inference pipeline for axon tracing algorithms to support large-scale brain mapping. Through three distinct input volumes of imagery, the distributed data inference pipeline was tested for scalability across varying compute nodes and dataset sizes. The results showed significant decreases in wall time, or time to final stitch, using 2 to 16 compute nodes each equipped with 2 Volta GPUs. Particularly noteworthy was the time improvement of the largest input volume (2048-cubed voxels) from its baseline time of 42:48:17 to 03:25:34, a 90.95% decrease. Additionally, the modular structure of the pipeline enabled an increased awareness and validation of operations as the output of each was saved independently before facilitating the next step in the pipeline. Further analysis can be done on other types of compute nodes available in a typical HPC environment.

The axon tracing algorithm [8] may be further explored and tuned to meet the demands of a particular dataset. The current crop size of the ML model is $128 \times 128 \times 64$ voxels. This was set to compare to the non-pipeline baselines, but may be changed by the user according to the raw image shape. In turn, a small enough crop size would allow the 256-cubed voxels data to be processed with more than 4 nodes. Microbatch size may also be increased to enable deeper parallel processing with DataParallel.

Future iterations of the pipeline could explore performance speed-ups in each step of the image processing, including preprocessing and postprocessing. Specifically, triples mode [16] can enhance our understanding of organizing and distributing data across nodes by specifically allocating the number of threads and processes per N nodes. A deeper analysis into the bottlenecks and time leaks may expose further areas of improvement as well as indicate the importance of each step. However, code optimization is a recommended first step, particularly in the implementation of split and stitch operations. For the prediction step, an interesting PyTorch module [17] may improve upon the use of DataParallel by distributing across multiple GPUs. Most literature on this area of parallelization concerns model training, which suggests the creation of a generalized pipeline for multiple model operations (training and inference).

Since HPC is a shared resource, a job array requesting N nodes may wait for a subset of N to become available and thus increase the wall time. As a job submission, this pipeline may benefit from a SLURM interface that prompts the user to select a particular N number of nodes based on the number of jobs currently submitted by other users. Per this method, the wall time will be longer than anticipated, but will attempt to be the shortest given the current number of available nodes. Another potential solution would be to experiment with stitching in real-time. In other words, combine the inference and stitch steps into a single job array submission that listens for complete inferences to begin stitching in real-time.

For the applicability of the inference speed-up, the pipeline should further explore input volumes that are diverse in shape and content density. A potential speed-up for less dense datasets is trimming or removing uninformative frames, such as those with a majority of background (zero intensity) voxels. The further consultation by experts in the intersecting areas of research (image processing, distributed systems, neuroscience, etc.) will enhance the development of these tools and generalize the use across domains.

Acknowledgment

The authors would like to acknowledge Adam Michaleas and the MIT Lincoln Laboratory Supercomputing Center (LLSC) for their support of high performance computing tasks.

References

[1] J. W. Lichtman, H. Pfister, and N. Shavit, "The big data challenges of connectomics," *Nature neuroscience*, vol. 17, no. 11, pp. 1448–1454, 2014.

[2] E. Plebani, N. P. Biscola, L. A. Havton, B. Rajwa, A. S. Shemonti, D. Jaffey, T. Powley, J. R. Keast, K.-H. Lu, and M. M. Dundar, "High-throughput segmentation of unmyelinated axons by deep learning," *Scientific Reports*, vol. 12, no. 1, pp. 1–16, 2022.

[3] F. Xu, Y. Shen, L. Ding, C.-Y. Yang, H. Tan, H. Wang, Q. Zhu, R. Xu, F. Wu, Y. Xiao *et al.*, "High-throughput mapping of a whole rhesus monkey brain at micrometer resolution," *Nature biotechnology*, vol. 39, no. 12, pp. 1521–1528, 2021.

[4] J. Park, J. Wang, W. Guan, L. Kamentsky, N. B. Evans, L. Gjesteby, D. Pollack, S. W. Choi, M. Snyder, D. Chavez *et al.*, "Integrated platform for multi-scale molecular imaging and phenotyping of the human brain," *bioRxiv*, 2022.

[5] T. Ku, J. Swaney, J.-Y. Park, A. Albanese, E. Murray, J. H. Cho, Y.-G. Park, V. Mangena, J. Chen, and K. Chung, "Multiplexed and scalable super-resolution imaging of three-dimensional protein localization in size-adjustable tissues," *Nature biotechnology*, vol. 34, no. 9, pp. 973–981, 2016.

[6] T. Ku, W. Guan, N. B. Evans, C. H. Sohn, A. Albanese, J.-G. Kim, M. P. Frosch, and K. Chung, "Elasticizing tissues for reversible shape transformation and accelerated molecular labeling," *Nature methods*, vol. 17, no. 6, pp. 609–613, 2020.

[7] T. Klinghoffer, P. Morales, Y.-G. Park, N. Evans, K. Chung, and L. J. Brattain, "Self-supervised feature extraction for 3d axon segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 978–979.

[8] D. Pollack, L. A. Gjesteby, M. Snyder, D. Chavez, L. Kamentsky, K. Chung, and L. J. Brattain, "Axon centerline detection using topologically-aware 3D U-Nets," in *International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, 2022.

[9] K. Lee, J. Zung, P. Li, V. Jain, and H. S. Seung, "Superhuman accuracy on the SNEMI3D connectomics challenge," *arXiv preprint arXiv:1706.00120*, 2017.

[10] S. Shit, J. C. Paetzold, A. Sekuboyina, I. Ezhov, A. Unger, A. Zhylka, J. P. Pluim, U. Bauer, and B. H. Menze, "cldice-a novel topology-preserving loss function for tubular structure segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 560–16 569.

[11] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60.

[12] Dataparallel documentation. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.DataParallel.html

[13] A. Collette, J. Tocknell, T. A. Caswell, D. Dale, U. K. Pedersen, A. Jelenak, A. Bedini, M. Raspaud, jialin, L. Hole, M. Brucher, M. Teichmann, G. A. Vaillant, jakirkham, K. Hinsen, P. de Buyl, A. Huebl, F. Rathgeber, T. Verstraelen, S. Sort, S. G. Ebner, smutch, M. Zwier, A. Lee, M. Brett, J. Kleinhenz, J. Bernhard, and J. Tyree. (2017, Mar.) h5py/h5py: 2.4.0. [Online]. Available: https://doi.org/10.5281/zenodo.400660

[14] Numpy documentation: array_split. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.array_split.html#numpy.array_split

[15] C. Byun, W. Arcand, D. Bestor, B. Bergeron, V. Gadepally, M. Houle, M. Hubbell, M. Jones, A. Klein, P. Michaleas *et al.*, "Node-based job scheduling for large scale simulations of short running jobs," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2021, pp. 1–7.

[16] A. Weinert, M. Brittain, N. Underhill, and C. Serres, "Benchmarking the processing of aircraft tracks with triples mode and self-scheduling," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, 2021, pp. 1–8.

[17] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala, "Pytorch distributed: Experiences on accelerating data parallel training," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 3005–3018, aug 2020. [Online]. Available: https://doi.org/10.14778/3415478.3415530