

# A Fast and Scalable Pipeline for Stain Normalization of Whole-Slide Images in Histopathology

Milos Stanisavljevic<sup>1</sup>, Andreea Anghel<sup>1</sup>, Nikolaos Papandreou<sup>1</sup>,  
Sonali Andani<sup>1</sup>, Pushpak Pati<sup>1</sup>, Jan Hendrik Rüschoff<sup>2</sup>, Peter Wild<sup>3</sup>,  
Maria Gabrani<sup>1</sup>, and Haralampos Pozidis<sup>1</sup>

<sup>1</sup> IBM Research – Zurich, 8803 Rüschlikon, Switzerland  
{ysm, aan, npo, sop, pus, mga, hap}@zurich.ibm.com

<sup>2</sup> Senckenberg Institute of Pathology, Universitätsklinikum Frankfurt,  
60590 Frankfurt am Main, Germany  
JanHendrik.Rueschoff@usz.ch

<sup>3</sup> Institute of Pathology and Molecular Pathology, UniversitätsSpital Zürich,  
8091 Zürich, Switzerland  
Peter.Wild@kgu.de

**Abstract.** Stain normalization is one of the main tasks in the processing pipeline of computer-aided diagnosis systems in modern digital pathology. Some of the challenges in this tasks are memory and runtime bottlenecks associated with large image datasets. In this work, we present a scalable and fast pipeline for stain normalization using a state-of-the-art unsupervised method based on stain-vector estimation. The proposed system supports single-node and distributed implementations. Based on a highly-optimized engine, our architecture enables high-speed and large-scale processing of high-magnification whole-slide images (WSI). We demonstrate the performance of the system using measurements from different datasets. Moreover, by using a novel pixel-sampling optimization we show lower processing time per image than the scanning time of ultrafast WSI scanners with the single-node implementation and additional 3.44 average speed-up with the 4-nodes distributed pipeline.

**Keywords:** Histopathological image processing, whole-slide images, stain normalization, distributed computing, color deconvolution

## 1 Introduction

In digital pathology, computer-aided diagnosis (CAD) has become an essential part of the clinical work with the advent of high-resolution whole-slide imaging technology. The fusion of machine learning (ML) based image analysis algorithms, and digitized histological slides are assisting the pathologists in terms of workload reduction, efficient decision support [23], [10] and interpretability of the results [21]. Given the vast amount of gigapixel-sized whole-slide imaging

data, and the need to accelerate the time-to-insight, there is an increasing demand to build automated and scalable pipelines for large-scale and fast image analysis.

Color normalization of stained tissue samples is one of the main preprocessing steps in whole-slide image (WSI) processing [24]. Despite the standardized staining protocols, variations in the staining results are very frequent due to differences in the staining parameters, e.g. antigen concentration and incubation time and temperature, different conditions between slide scanners, etc. [21]. Such color/intensity variations can adversely affect the performance and accuracy of the CAD systems. Therefore, stain normalization techniques have been proposed to generate images with a standardized appearance of the different stains [17, 15, 11, 8, 9, 2, 1, 7].

In this work, we use the Macenko method [11] to implement a high-performance stain normalization system. The algorithm does not involve intermediate steps that require training of model parameters and is thus computationally less expensive. Our stain normalization system is based on an optimized multi-core implementation of the singular-value decomposition-based method (SVD) in [11]. In addition, to support the processing of high-magnification images, where the size of the image may not fit in the CPU memory (e.g., a 40X magnification WSI of 160k×80k pixels corresponds to 37.5 GB of data in RGB), we devise an iterative multi-batch implementation. Furthermore, we design 2 system flavors: single-node and distributed multi-node versions. The latter offers a scalable solution that enables large-scale and high-speed processing of high-resolution WSIs using a cluster of nodes with multi-core CPUs. Finally, our implementation supports multiple image formats (e.g., .svs, .tiff, .ndpi) which enables the evaluation of stain normalization on datasets generated by different scanners.

Our contributions are the four-fold as follows:

- (a) A high-performance implementation of the Macenko algorithm [11] that enables processing of gigapixel WSI (magnification 40X and beyond);
- (b) A distributed architecture that uses computing power of a cluster to further accelerate the stain normalization workload;
- (c) A pipeline that supports multiple execution modes, i.e., single- or multi-node execution, depending on the image size and the system resources, e.g., available RAM, cluster nodes (machines);
- (d) An evaluation of the proposed system on WSI datasets generated by different scanning systems thus having different image formats and characteristics.

In the next sections, we present the architecture and implementation aspects of our novel stain normalization system. We discuss the optimization steps and the role of the various parameters in the runtime and accuracy of the algorithm.

## 2 Stain Normalization of Whole-Slide Images

The stain normalization method presented in [11] belongs to the class of unsupervised normalization methods. The algorithm estimates first the hematoxylin

and eosin (H&E) stain vectors of the WSI of interest by using an SVD approach on the non-background pixels. Second, the algorithm applies a correction to account for the intensity variations due to noise. The algorithm is based on the principle that the color of each pixel (RGB channels) is a linear combination of the two stain vectors which are unknown and need to be estimated.

As a reference implementation of the Macenko algorithm [11] we use a publicly available MATLAB implementation [19]. We outline the algorithmic steps in Algorithm 1. Additional details are available in [22].

---

**Algorithm 1** SVD method for obtaining the stain vectors

---

- 1: Convert RGB to optical density (OD)
  - 2: Remove data with negligible optical density
  - 3: Calculate SVD of the OD tuples and use the largest 2 values to create SVD plane
  - 4: Project data onto the plane, normalize to unit length
  - 5: Calculate the angle  $\phi$  of each point with respect to the 1<sup>st</sup> (or 2<sup>nd</sup>) SVD direction
  - 6: Find the robust extremes ( $\alpha^{th}$  and  $(100 - \alpha)^{th}$  percentiles) of the angle  $\phi$
  - 7: Find the projection of the extreme values back to OD space
  - 8: Use this projection as optical density matrix (ODM, see [17])
  - 9: Calculate the individual stain concentrations ( $C_h$  and  $C_e$ ) using the inverse of ODM
  - 10: Find the robust max. ( $(100 - \alpha)^{th}$  percentile) of the individual stain concentrations
  - 11: Normalize and transform concentrations to OD space and then back to RGB
- 

### 3 High-Performance Stain Normalization Architecture

#### 3.1 Optimized Multi-Core Architecture

We develop two optimized single-node implementations: (a) single-batch and (b) multi-batch. The single-batch implementation is intended to minimize the processing time on a single-node, while the multi-batch implementation enables processing of 40X WSI when their size is larger than RAM of a single node (Table in Fig. 4(c)). Both implementations follow the steps shown in Algorithm 1. Due to multiple optimizations, we reorganize the steps as in Fig. 1.



**Fig. 1.** Mapping of the Macenko Algorithm 1 steps to our optimized implementation

To enable single-batch processing of 40X images on state-of-the-art machines with 64GB of RAM, only RGB pixel values are permanently stored in memory. Steps 1-2 of Algorithm 1 are executed multiple times, in the processing blocks A,

$B$  and  $D$  in Fig. 1, because these blocks operate on non-background pixels in OD space. *Step 1* is executed also in block  $E$ , because this block requires all pixels in OD space. *Step 1* is performed using a 256-entry lookup-table that greatly speeds up the logarithm calculation. Moreover, *Step 2* removes the background pixels which significantly reduces both the processing and the memory load. In the multi-batch implementation, batches of RGB pixel values are only temporarily stored in RAM and are read from the file system in the blocks  $A$ ,  $B$ ,  $D$ , and  $E$ .

To speed up the CPU processing we perform the following optimizations:

(a) In block  $A$ , during SVD calculation, the covariance matrix is calculated using the property that the element  $(i, j)$  of the matrix,  $\Sigma_{ij} = \frac{1}{N^2} (\sum_p x_{p,i} x_{p,j} - \sum_p x_{p,i} \sum_p x_{p,j})$ , requires only the sums of OD components.

(b) In blocks  $B$  and  $D$ , which are benchmarked as the most time-consuming steps, partial sorting is performed to find the percentiles from *Steps 6* and *10*. This partial sorting runs 2-3x faster compared to full sorting. In the multi-batch implementation, individual batches are partially sorted and then combined using an optimized merging function to calculate the global robust extremes.

(c) For the exponential function in the processing block  $B$ , we use the fast exponentiation library [4] since it performs 5-10x faster compared to the corresponding function in the standard C library.

(d) Since the processing blocks  $A$ ,  $B$ ,  $D$ , and  $E$  perform many independent operations on individual pixels, their execution is parallelized across all available CPU threads using the OpenMP library [13].

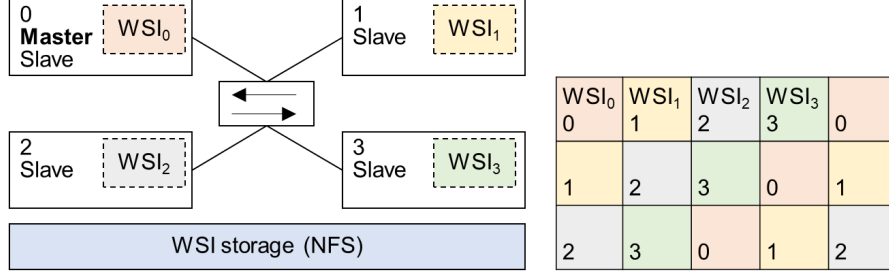
Given that the processing blocks  $B$  and  $D$  are the most time-consuming due to the difficulty of parallelizing the sorting operation, we propose another optimization that is using a Monte Carlo sampling technique [5]. In this method, a sample of non-background pixels is randomly chosen from the set of all non-background pixels in order to estimate the required robust extremes from *Steps 6* and *10*. Despite different methods for estimating the population percentiles [18], estimating the variance of the percentile estimates is unreliable, thus making the analytical estimation of the required sample size difficult [3]. Therefore, we derive the optimal sample size based on empirical results in Section 5.

### 3.2 Distributed Architecture

We present a novel distributed implementation of the Macenko algorithm. This implementation is useful when the WSI size is significantly larger than the RAM of a single node. The distributed solution uses all the optimizations presented in Section 3.1. We provide 2 multi-node implementations: (1) single-batch (the local image partition is fully read once) and (2) multi-batch (the local image is split into batches which temporarily reside in RAM and read multiple times when needed). For inter-node communication we use the MPI library [12].

Fig. 2 shows an overview of the multi-node setup. The WSI typically resides in a file system shared across all nodes. We split the WSI into partitions  $WSI_i$ ,  $i \in \{0, 1, 2, 3\}$ , and assign each partition to a separate node. Each node reads only its assigned partition. All nodes run the optimized Macenko algorithm on their partitions in parallel. Some processing steps need synchronization across

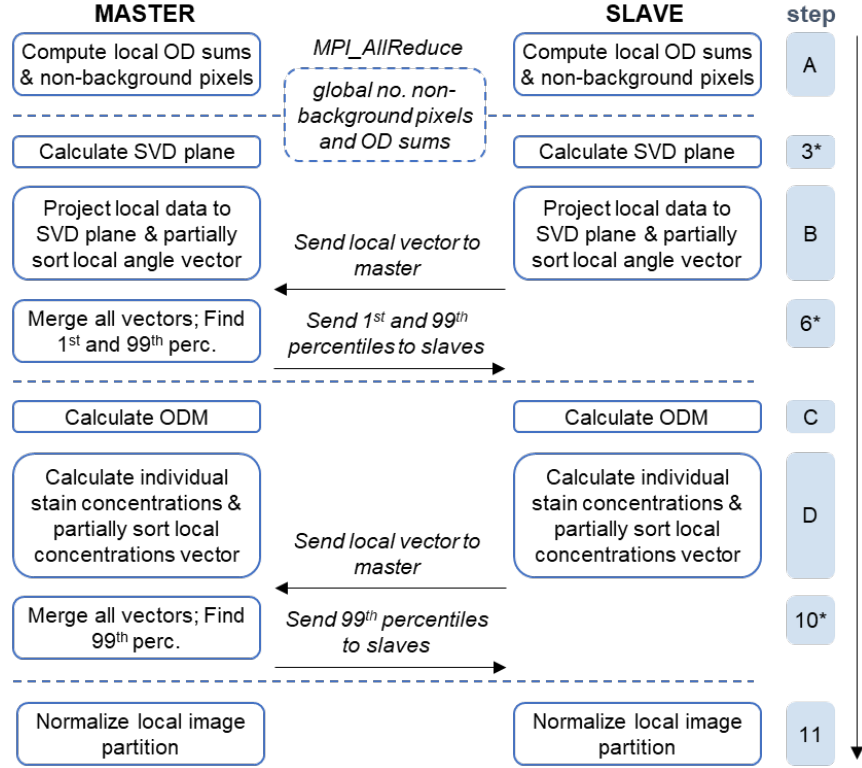
nodes, e.g. *Steps* 6 and 10 in Algorithm 1. Thus one node needs to aggregate the relevant pixel data and compute, e.g. the global robust percentiles. We call this node *master*. Fig. 2 shows a cluster of 4 nodes interconnected via a network, each node being assigned a part of the image. All nodes are slaves and node 0 is also master. We use this cluster setup for the experiments run in Section 5.



**Fig. 2.** Cluster setup and image partitioning scheme.

To describe the image partitioning, we define the partitioning element granularity as being a tile (stripe) for the single-batch implementation and a batch (e.g., a set of tiles/stripes) for the multi-batch implementation. The tile (stripe) is a rectangular contiguous part of the image, e.g., a tile is a 256x256-pixel image region. Let  $N$  be the number of nodes (indexed from 0). The image is composed of elements (tiles or batches) that form a grid with  $C$  columns and  $L$  rows. The total number of elements  $E$  is equal to  $C \times L$ . Each element  $e$  can be uniquely described by 2 coordinates  $c$  and  $l$ , where  $c \in \{0, \dots, C-1\}$  and  $l \in \{0, \dots, L-1\}$ . Given  $c$  and  $l$ , we compute the element index as  $e(c, l) = l \cdot C + c$ . The partitioning across nodes is performed by assigning element  $e(c, l)$  to node  $[e(c, l) \bmod N]$ . Fig. 2 shows a partitioning example, where  $N = 4$ ,  $C = 5$  and  $L = 3$ . This partitioning scheme is preferred over assigning contiguous sets of elements to each node in order to reduce the node imbalance due to input content. For example for 2 nodes and an image in which the top half is background, the contiguous scheme would assign an empty image to node 0. In this case, there will be a large processing imbalance between the 2 nodes, impacting the scalability of the distributed system. For the multi-node single-batch implementation, given  $t_H$  the tile height,  $t_W$  the tile width,  $H$  the WSI height and  $W$  the WSI width, all variables expressed in pixels,  $C = \lceil W/t_W \rceil$  and  $L = \lceil H/t_H \rceil$ . For the multi-node multi-batch implementation, the user defines a batch size  $B_{size}$  as being the maximum number of pixels that can reside in RAM at any given time. Assuming square batches, then a batch will have the width and height equal to  $\lfloor \sqrt{B_{size}} \rfloor$ , which implies that  $C = \lceil W/\lfloor \sqrt{B_{size}} \rfloor \rceil$  and  $L = \lceil H/\lfloor \sqrt{B_{size}} \rfloor \rceil$ .

Fig. 3 shows the communication flow across nodes at different algorithm stages. First, each node reads its partition, runs *Steps* 1-2 of Algorithm 1 and computes the local OD sums and number of non-background pixels. Next, we

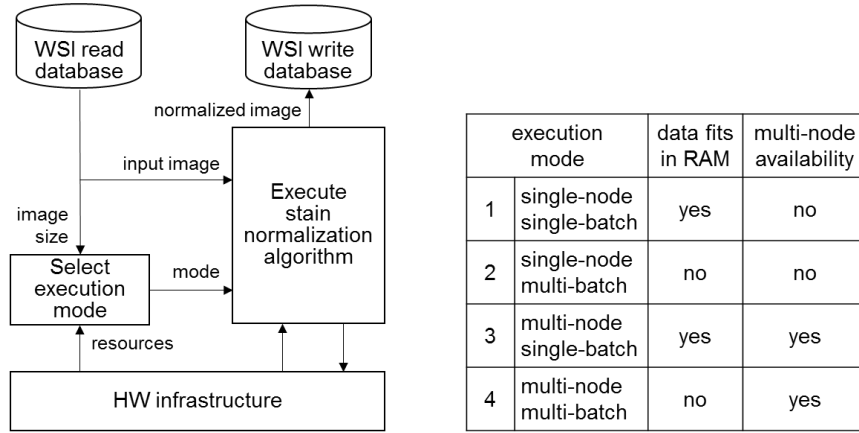


**Fig. 3.** Communication stages across nodes at different stages of the stain normalization algorithm.

run an MPI reduction phase to compute on all nodes the global number of non-background pixels and OD sums. Then, each node runs *Steps 3-5* and sends its local vectors of angles to the master, where the global 1<sup>st</sup> and 99<sup>th</sup> percentiles of the projected angles are computed. The master then sends these percentiles to all slaves so that they can run *Steps 7-9*. The master computes the global 99<sup>th</sup> percentiles of the stain concentrations based on the local concentration vectors sent by the slaves. The global percentiles are sent to all slaves which are used for normalization. Finally each node writes its image partition to a file.

### 3.3 Mode Selection Pipeline

Fig. 4 shows the pipeline architecture for implementing mode selection depending on the image size and the system resources. Given the input image size, which depends on the image format and the targeted magnification, e.g., 10X, 40X, etc., and the system resources in terms of number of available nodes and main memory capacity per node, the optimal execution mode is selected. For example, if the image fits in the memory of a single node, then *mode 1* in the table shown



**Fig. 4.** High-level diagram of the mode selection pipeline depending on image size and hardware resources. *Data* (in table) refers to either the full WSI or the image partition assigned to the node.

in Fig. 4(c) is selected. Otherwise, *mode 2* is selected and the stain normalization engine will process the image in batches. If a cluster of nodes is available, *mode 3* or *4* will be selected depending on whether the data partitioned among the different nodes fits in RAM or not, respectively.

## 4 Whole-Slide Image Datasets

To analyze the performance and scalability of our pipeline, we used H&E-stained WSIs from 4 datasets. The first one is part of TUPAC MICCAI 2016 [20] and provides breast WSIs for prediction of tumor and proliferation scores. These WSIs are in Aperio format, single-file pyramidal tiled TIFF (.svs), with JPEG compression scheme. The CAMELYON16 dataset [6] is part of the ISBI challenge on cancer metastasis detection in lymph node. These slides are in Philips format, single-file pyramidal tiled TIFF or BigTIFF (.tif) with non-standard metadata and JPEG compression scheme. The remaining 2 datasets are proprietary but we used them to test the flexibility of our pipeline to different WSI formats. One dataset provides slides in Ventana format, single-file pyramidal tiled BigTIFF with non-standard metadata. The other dataset contains slides in Hamamatsu format, single-file TIFF-like format (.ndpi) with proprietary metadata. All the slides include 2.5X, 10X and 40X magnifications, except for the fourth dataset that includes 2.5X and 10X magnifications only. A de facto community standard for reading various WSI formats is OpenSlide C library that provides a simple interface to WSIs [14]. However, we wrote the proprietary functions that rely on the existing API from libTIFF and BigTIFF standard libraries for reading and writing of supported WSI formats that allow significantly faster read times as shown in Table 1 for various image formats and magnifications. In addition, our

read functions have lower memory requirements compared to OpenSlide library functions.

**Table 1.** Image read speed-up of our implementation compared to OpenSlide

Scanner type	10X speed-up	40X speed-up
Aperio	4.7	4.7
Philips	4.9	<b>22.2</b>
Ventana	6.4	4.6
Hamamatsu	14.3	N.A.

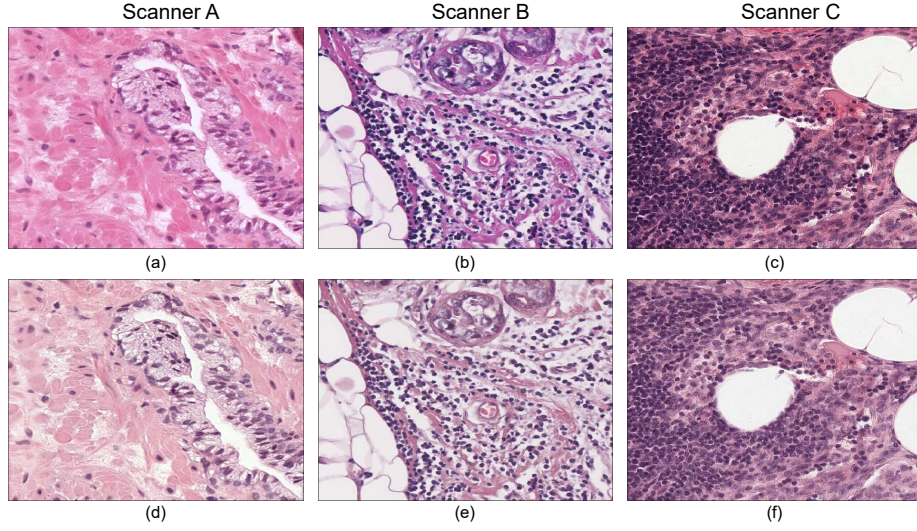
## 5 Experimental Results

**40X WSI Processing.** Fig. 5 shows three images generated by different scanners, which we will further refer to as Scanner A, B and C, respectively. The top row shows the original images and the bottom row shows the normalized versions after running the normalization pipeline of Section 3. Each panel in Fig. 5 corresponds to a 1024x1024-pixel region of the image at 40X magnification. While the input images (top row) show significant color variation, the normalized ones show more uniform color contrast as a result of the normalization process.

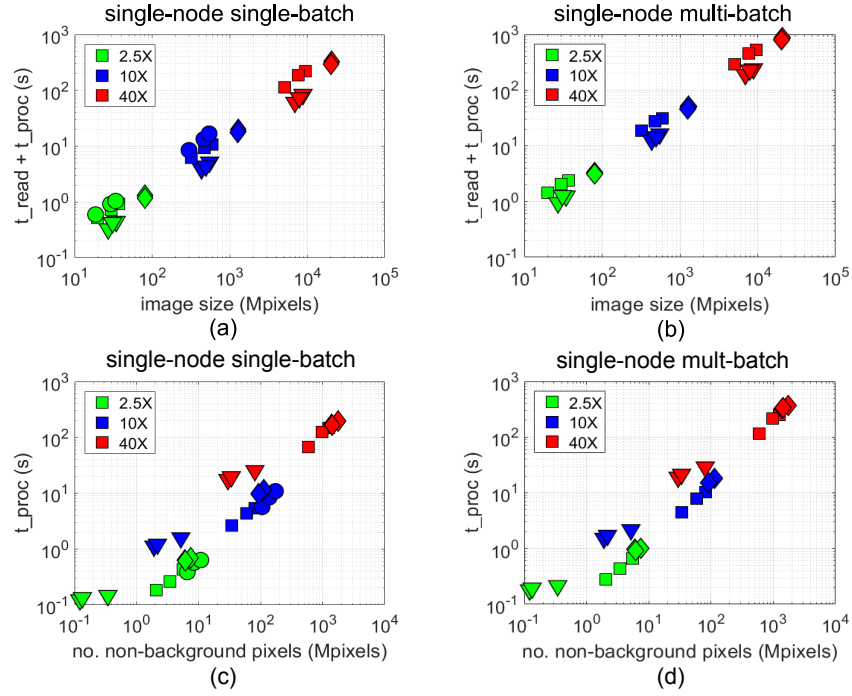
**Single-Node Results.** Figs. 6(a)-(d) present measurements of the processing time of the single-node system. Figs. 6(a)-(b) show the total processing time, including the time to read the images, as a function of the image size for the single- and multi-batch implementations in double-logarithmic scale. The different colors correspond to images in 2.5X, 10X, and 40X magnification, while the different markers correspond to the different datasets. The multi-batch implementation exhibits a moderate time increase that is mainly attributed to the multiple image reading overhead. A number of 4 batches has been used in all single-node multi-batch measurements. Figs. 6(c)-(d) show the corresponding processing time only as a function of the number of non-background pixels. For each magnification, the measurements scale almost linearly with the number of non-background pixels. The processing time shows an offset for higher magnifications due to the initial memory allocation and final image normalization steps that are performed on the whole image.

**Single-Node vs. Reference Results.** Table 2 reports the average execution time over all images for all datasets in 10X magnification using the single-node implementation described in Section 3.1, compared with the original MATLAB code in [19] and reference C++ implementation based on it using OpenSlide library for reading of the images. All measurements have been collected on a single node with a 10-core Intel® i7-6950X CPU at 3GHz and 64GB of RAM. Our implementation achieves a speed-up factor larger than 8 for 10X images.





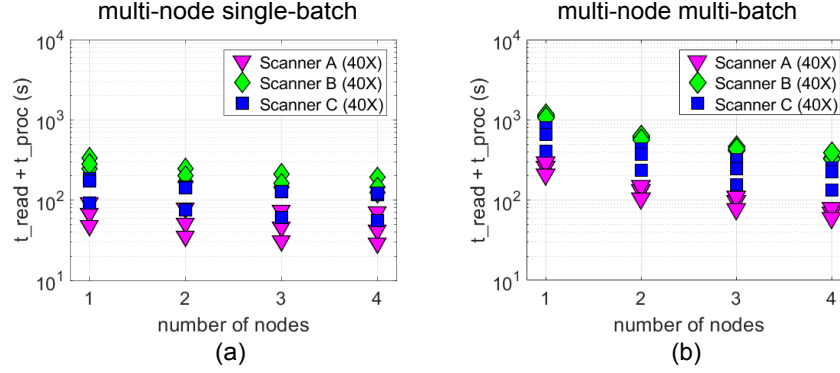
**Fig. 5.** Input images from different datasets and scanners (a)-(c) and normalized output images (d)-(f) after applying the stain normalization pipeline



**Fig. 6.** Experimental results showing the performance of the proposed stain normalization pipeline on WSI from different datasets and magnification factors

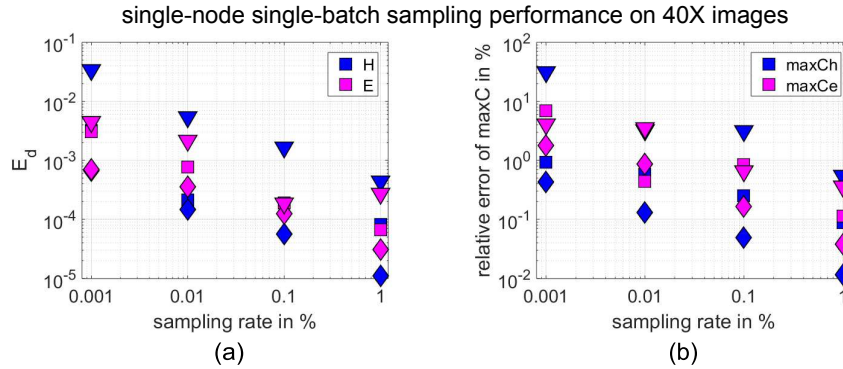
**Table 2.** MATLAB vs. single-node implementations (averaged for 10X images)

Implementation	$t_{\text{proc}} + t_{\text{read}}$ (s)	Gain
MATLAB [13]	147	1
reference w/ OpenSlide	124	1.19
single-batch	18	8.17
multi-batch	37	3.97

**Fig. 7.** Performance of distributed implementation based on the number of nodes for different image types and 40X magnification.

**Multi-Node Results.** Figs. 7(a)-(b) show the scalability results of the multi-node single-batch and multi-batch implementations, respectively. Measurements have been collected on a cluster of nodes with an 8-core Intel® Xeon® E5-2630v3 CPU and 64GB of RAM. We show aggregated processing and read time as a function of the number of nodes for 40X magnification images. Each point and color represent an image and a dataset, respectively. For the single-batch implementation the speed-ups when increasing the number of processing nodes compared to the single-node measurements are as follows: 1.70X (2 nodes), 2.29X (3 nodes) and 2.80X (4 nodes). For the multi-batch implementation we measure the following average speed-ups compared to the single-node multi-batch measurements: 1.87X (2 nodes), 2.60X (3 nodes) and 3.30X (4 nodes). The reasons for the sub-linear scaling are the communication overhead of *Blocks* B and D in Fig 3, and aggregation of intermediate results (merging of the vectors) in *Blocks* 6\* and 10\* in Fig 3.

**Single-Node Pixel Sampling Results.** Figs. 8(a)-(b) show the Euclidean distance ( $E_d$ ) of ODM and the relative error of the robust maximum of the individual stain concentrations ( $\max C_h$  and  $\max C_e$ ), respectively, between the sampling and no-sampling results. The measured  $E_d$  values of the individual stains are almost negligible compared to typical values such as the ones reported in Table 1 of [1] with a sampling rate of non-background pixels as low as 0.01%. Similarly, the relative error of the robust maximum estimation drops below 1%



**Fig. 8.** Euclidean distance of ODM and the relative error of the robust maximum of the individual stain concentrations for various sampling rates.

**Table 3.** Processing time for different sampling rates, single-node single-batch implementation (40X images)

Dataset	0.001%	0.01%	0.1%	1%	no sampling
Scanner A	15.6	15.7	15.7	16.2	20.8
Scanner B	42.8	42.9	43.3	44.6	177.7
Scanner C	15.7	15.8	16.0	16.9	113.0

for a sampling rate of 1%. The higher sampling rate value in case of robust maximum estimation is required to offset the error aggregation from ODM estimation. Nevertheless, the increase in total processing time with respect to the sampling rate is negligible even for rates as large as 1%, as shown in Table 3. The average processing time (without read time) for different sampling rates and for the single-node single-batch implementation (no-sampling) is shown in Table 3. Thanks to the sampling method, the overall processing time is reduced by a factor of 1.3-9x depending on the image type, for 40X magnification. This effectively reduces the processing time to normalization time yielding this implementation as fast as methods that apply a fixed reference normalization template regardless of the input image [16].

**Best Single-Batch Results.** Table 4 shows the average speed-ups across all 40X magnification datasets of the single-node single-batch (with 1% sampling), four-node single-batch (with and without sampling) when compared to the single-node single-batch implementation without sampling. By combining the sampling technique with the distributed pipeline we attain an average speed-up of 3.44 and 5.19 vs. the single-node with and without sampling, respectively.

**Table 4.** Single-batch implementations (averaged for 40X images)

Implementation	Gain
single-node	1
single-node w/ sampling	1.51
four-node	2.8
four-node w/ sampling	5.19

## 6 Conclusions

We built a fast and scalable pipeline to enable large-scale stain normalization of high-resolution histopathological whole-slide images. Our pipeline uses a highly optimized low-level engine that performs the required image processing functions and is based on a distributed computing architecture that is scalable in both image size and number of computing nodes. The presented pipeline tackles the memory and runtime bottlenecks of high-magnification images and enables the preprocessing of large datasets, which is a critical prerequisite for any ML framework applied to biomedical images. Our next steps involve: (a) performance evaluation of ML frameworks applied to stain normalized images, (b) automation of the pipeline mode selection, and (c) automated batch distribution based on the number of non-background pixels for load balancing across nodes.

## References

1. Alsubaie, N., Trahearn, N., Raza, S.E.A., Snead, D., Rajpoot, N.M.: Stain deconvolution using statistical analysis of multi-resolution stain colour representation. *PLOS ONE* **12**(1), e0169875 (Jan 2017). <https://doi.org/10.1371/journal.pone.0169875>
2. Bejnordi, B.E., Litjens, G., Timofeeva, N., Otte-Hiller, I., Homeyer, A., Karssemeijer, N., van der Laak, J.A.: Stain specific standardization of whole-slide histopathological images. *IEEE Transactions on Medical Imaging* **35**(2), 404–415 (Feb 2016)
3. Brown, M.B., Wolfe, R.A.: Estimation of the variance of percentile estimates. *Computational Statistics & Data Analysis* **1**, 167–174 (1983)
4. Fast approximate function of exponential function exp and log. [Online] Available: <https://github.com/herumi/fmath>
5. Harrison, R.L.: Introduction to Monte Carlo Simulation. In: Granja, C., Leroy, C. (eds.) *American Institute of Physics Conference Series*. American Institute of Physics Conference Series, vol. 1204, pp. 17–21 (Jan 2010). <https://doi.org/10.1063/1.3295638>
6. ISBI challenge on cancer metastasis detection in lymph node. [Online] Available: <https://camelyon16.grand-challenge.org/data/>
7. Janowczyk, A., Basavanthally, A., Madabhushi, A.: Stain normalization using sparse autoencoders (stanosa): Application to digital pathology. *Computerized Medical Imaging and Graphics* **57**, 50 – 61 (Apr 2017)
8. Khan, A.M., Rajpoot, N., Treanor, D., Magee, D.: A nonlinear mapping approach to stain normalization in digital histopathology images using image-specific color deconvolution. *IEEE Transactions on Biomedical Engineering* **61**(6), 17291738 (June 2014)
9. Li, X., Plataniotis, K.N.: A complete color normalization approach to histopathology images using color cues computed from saturation-weighted statistics. *IEEE Transactions on Biomedical Engineering* **62**(7), 1862–1873 (July 2015)
10. Litjens, G., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., van der Laak, J., van Ginneken, B., Snchez, C.I.: A survey on deep learning in medical image analysis. *Medical Image Analysis* **42**, 60 – 88 (2017)
11. Macenko, M., Niethammer, M., Marron, J.S., Borland, D., Woosley, J.T., Guan, X., Schmitt, C., Thomas, N.E.: A method for normalizing histology slides for quantitative analysis. In: 2009 IEEE International Symposium on Biomedical Imaging. pp. 1107–1110 (2009)
12. Open MPI: Open Source High Performance Computing. [Online] Available: <https://www.open-mpi.org/software/ompi/v3.1/>
13. OpenMP 4.0 Specifications. [Online] Available: <https://www.openmp.org/specifications/>
14. OpenSlide is a C library that provides a simple interface to read whole-slide images. [Online] Available: <https://openslide.org/>
15. Rabinovich, A., Agarwal, S., Laris, C., Price, J., Belongie, S.: Unsupervised color decomposition of histologically stained tissue samples. In: *Adv. Neural Inf. Process. Syst.* pp. 667–674 (2003)
16. Reinhard, E., Adhikhmin, M., Gooch, B., Shirley, P.: Color transfer between images. *IEEE Computer Graphics and Applications* **21**(5), 34–41 (Sep 2001). <https://doi.org/10.1109/38.946629>
17. Ruifrok, A.C., Johnston, D.A.: Quantification of histochemical staining by color deconvolution. *Anal. Quant. Cytol. Histol.* **23**(4), 291–9 (August 2001)

18. Schoonjans, F., De Bacquer, D., Schmid, P.: Estimation of population percentiles. *Epidemiology (Cambridge, Mass.)* **22**(5), 750 (2011)
19. Staining unmixing and normalization.  
[Online] Available: <https://github.com/mitkovetta/staining-normalization>:
20. Tumor Proliferation Assessment Challenge 2016, TUPAC16 - MICCAI Grand Challenge. [Online] Available: <http://tupac.tue-image.nl/node/3>:
21. Veta, M., Pluim, J.P.W., van Diest, P.J., Viergever, M.A.: Breast cancer histopathology image analysis: A review. *IEEE Transactions on Biomedical Engineering* **61**(5), 1400–1411 (May 2014)
22. Vink, J.P., Leeuwen, M.B.V., Deurzen, C.H.M.V., Haan, G.D.: Efficient nucleus detector in histopathology images. *Journal of microscopy* **249**(2), 124–135 (2013)
23. Wernick, M.N., Yang, Y., Brankov, J.G., Yourganov, G., Strother, S.C.: Machine learning in medical imaging. *IEEE Signal Processing Magazine* **27**(4), 25–38 (July 2010)
24. Zerhouni, E., Lnyi, D., Viana, M., Gabrani, M.: Wide residual networks for mitosis detection. In: 2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017). pp. 924–928 (2017)