Published in final edited form as:

Distrib Parallel Databases. 2019 June ; 37(2): 251–272. doi:10.1007/s10619-018-7237-1.

MaReIA: A Cloud MapReduce Based High Performance Whole Slide Image Analysis Framework

Hoang Vo,
Department of Computer Science, Department of Biomedical Informatics, Stony Brook University, Stony Brook, NY

Jun Kong,
Department of Biomedical Informatics, Emory University, Atlanta, GA

Dejun Teng,
Department of Computer Science and Engineering, Ohio State University, Columbus, OH

Yanhui Liang,
Department of Computer Science, Department of Biomedical Informatics, Stony Brook University, Stony Brook, NY

Ablimit Aji,
Hewlett Packard Labs, Palo Alto, CA

George Teodoro, and
Department of Computer Science, University of Brasília, Brasília, DF, Brazil

Fusheng Wang
Department of Computer Science, Department of Biomedical Informatics, Stony Brook University, Stony Brook, NY

Abstract

Recent advancements in systematic analysis of high resolution whole slide images have increase efficiency of diagnosis, prognosis and prediction of cancer and important diseases. Due to the enormous sizes and dimensions of whole slide images, the analysis requires extensive computing resources which are not commonly available. Images have to be tiled for processing due to computer memory limitations, which lead to inaccurate results due to the ignorance of boundary crossing objects. Thus, we propose a generic and highly scalable cloud-based image analysis framework for whole slide images. The framework enables parallelized integration of image analysis steps, such as segmentation and aggregation of micro-structures in a single pipeline, and generation of final objects manageable by databases. The core concept relies on the abstraction of objects in whole slide images as different classes of spatial geometries, which in turn can be handled as text based records in MapReduce. The framework applies an overlapping partitioning scheme on images, and provides parallelization of tiling and image segmentation based on MapReduce architecture. It further provides robust object normalization, graceful handling of boundary objects with an efficient spatial indexing based matching method to generate accurate

results. Our experiments on Amazon EMR show that *MaReIA* is highly scalable, generic and extremely cost effective by benchmark tests.

Keywords

Whole Slide Images; Spatial Application; Pathology Image Analysis; MapReduce; Cloud Computing

1 Introduction

In the recent decade, applications of high-resolution microscopy imaging are gaining influential role in quantitative characterization of biological structures, discovery of insights on disease mechanisms, facilitation and development of exams, therapies and procedures. Improvement in pathology analytical imaging, supported by the emergence of cost-effective digital scanners, has prompted large-scale quantitative and integrative studies focusing on high throughput analysis of imaging features and annotations [12,15,19,20,11,21].

However, while high resolution scanners can produce whole slide images of up to 100K x 100K pixels rapidly, processing such images is very complex and highly computational intensive. For instances, histopathological operations, such as nuclear segmentation, generate millions of nuclear boundaries on average for each single image. Therefore, it may take weeks to process images with an ordinary desktop computer for a typical research study with hundreds of images. A regular desktop computer or server would require weeks to months for processing of these images, since each typical study has hundreds to thousands images.

Due to these computational limits of single machine caused by high dimensions of whole slide images, traditional image analysis techniques cannot be applied directly. Furthermore, the size of individual images might often be too large to fit in the memory of single machine. The direct application of these algorithms is only executable on super servers with extremely high memory availability, and would still be limited by extremely long execution time for each image. In particular, the majority of image processing algorithms are computeand data-intensive. The fastest algorithms, those with high polynomial complexity, might consume an extremely long amount of time to traverse the entire image multiple times for simply object segmentation. In contrast, medical doctors, analysts, pathologist and researchers would be discouraged by such long processing time and delay.

Whole slide image analysis with high throughput demands extensive computing resources such as high performance computing clusters [24] or grid computing [15]. The former approach is expensive and often inaccessible, and the grid approach suffers from major data transportation cost over the Internet. To handle large scale whole slide images, traditional approach partitions images into series of fixed sized tiles. Objects intersecting the tile boundaries, however, are often discarded, leading to inaccurate results.

The issue involving handling such boundary crossing objects in image parallel-processing is denoted as boundary object handling. If each processor task only handles a fragment of an

image, there is a high probability that some object span multiple regions of the image; the results need to be combined or aggregated. In-house computing infrastructures have limited computing capacity to process large batches of such images. Particularly, traditional algorithms often ignore boundary objects. While our studies have shown that there are up to 0.5–2 percent nuclear objects for a 4K x 4K tiling of brain tumor images at 20X resolution, processing tiles of lower dimension can be more efficient in utilizing computing resources, since smaller tile dimension would statistically decrease the number of objects to be processed per tile. Thus, achieving higher computational efficiency would potentially create more inaccurate results.

On the other hand, cloud computing solutions, such as Amazon, Google and Microsoft, offer computational power at low cost. The architecture layer allows both horizontal scalability and flexibility to use different algorithms on a diverse set of computing architectures, such as Apache MapReduce [2], Spark [3], Storm [4]. Our solution, MaReIA, based on our previous work [25], takes the advantage of any cloud computing environments supporting MapReduce or similar architecture, but it can also be deployed with ease on other cloud computing platforms, such as Spark, or in a in-house cluster environment.

The second challenge in parallel distributed computing is data being distributed over multiple machines or nodes in clusters; thus if a machine or node fails due to hardware issues during computation, it is not feasible to restart the entire work flow from the beginning. A potential solution to the problem is a fault-tolerant feature built in several cloud computing frameworks.

Cloud computing relies on a distributed and interconnected network of computers such as commodity clusters and Amazon EC2 to provide computation, storage, and resource management capabilities elastically in large scale.

MapReduce [13] is a popular computing paradigm for cloud computing, with Apache Hadoop is its most common open source implementation. MapReduce gained its popularity in biomedical research due to its transparent parallelism, scalability, fault tolerance, load balancing, and most importantly, simplicity in programming [8,9]. Little modification is needed to parallelize task across a cluster of machines.

Newer frameworks, such as Spark and other cloud computing architectures rely on underlying principles similar to Map and Reduce. They further leverage the power of MapReduce by using memory resources more effectively and optimizing I/O operations. However, for compute-intensive operation such as image processing of wholeslide images, MapReduce has comparable performance to newer in-memory processing approaches such as Spark [16].

More notably, HIPI, an image processing by Sweeny et al is a MapReduce based image processing where each task handles a group of small images [23]. These images can be perfectly processed in parallel due to their small size and their features are extracted independently. Markonis et al [22] proposed a MapReduce based system in image analysis for machine learning and parameter optimization task in general medical image analysis. However, it assumes and operates only on objects in a pre-segmented dataset. OpenIMAJ

developed by Hare et al offered a Java-based approach with limited MapReduce support for pixel-level operation and segmentation on original raster images. Its MapReduce support is again primarily for feature extraction and machine learning operation parallelization [17].

In the domain of high performance image recognition, Spark based Deep Learning frameworks operate on large numbers of small to medium images instead; its common applications are distributed cross validation [7]. However, they often lack a systematic approach to handle boundary crossing objects in larger scale images. Furthermore, while these frameworks leverages the power of in memory processing for iterative tasks in deep learning; therefore, they do not scale very well in scenarios requiring high system throughput.

In this paper, we propose a highly scalable and cost effective MapReduce based high performance image analysis framework for whole slide image processing, using a representative nuclear segmentation algorithm [18] and spatial vectorization of objects. The architecture can be flexibly adjusted to Spark and other distributed frameworks. Other segmentation algorithms for nuclei, blood vessel and various types of objects can be incorporated into the framework with little effort. Furthermore, multiple types of objects can be segmented with a single pass through the data. In particular, we propose an overlapping partitioning scheme providing tile level parallelism for MapReduce to support accurate segmentation and eliminate the boundary crossing object problem. This problem caused by partitioning in distributed processing is handled by our generic spatial indexing based matching method, which performs a spatial self-join operation on the data. The framework is implemented for both commodity clusters and commercial clouds including Amazon EMR, part of the Amazon Web Services (AWS) cloud. Our performance study involving benchmarks showed the robustness and scalability on large data sets.

The organization is as following; we first provide assumptions and background information for MapReduce architecture and Whole slide image analysis in Sec. 2. Our methods are described in Sec. 3, followed by our experiments and conclusion in Sec. 4 and Sec. 5, respectively.

2 Background

In this section, we explain and highlight the core principles of MapReduce architecture, followed by features of pathology images affecting our system design. MapReduce is a parallel computing framework invented by Google for large scale data analytics [13]. MapReduce not only provides a simplified programming model convenient for parallel application development, but also offers an infrastructure for reliable distributed processing. MapReduce programming model consists of two major phases operating on key-value records: i) a map function splitting the input problem into smaller subproblems identified by keys and distributes them to workers, and ii) a reduce function collecting answers of sub-problems and combining them to form the output. MapReduce provides distributed parallel processing of both map and reduce tasks on both process and node levels. Records are represented as *(key, value)* pairs for processing, Notably, the shuffling phase between map and reduce steps which re-groups result records from mapping phase into records sorted by

the reduce key; all key-value pairs with the same key are guaranteed to be logically grouped and processed by a single process.

Hadoop Distributed File System (HDFS) is a distributed and scalable file system for the Hadoop MapReduce framework. It provides large block based storage with distribution replication, offering high availability and fault-tolerance. Many cloud and service based storages such as Amazon Simple Storage Service (Amazon S3) support HDFS based storage.

Digital whole slide images (WSI) are frequently stored in a pyramid structure consisting of multiple images at different resolutions [26]. The base level contains rasterized data at the highest resolution. Most vendor storage formats support random access of regions. Due to enormous size of each image, tasks such as displaying the entire image and performing analysis on the image are limited by memory resource and monitor size. Thus, a common practice is to partition or divide whole slide image into smaller tiled images such that a tile can fit into computer memory for processing. While tiles partitioned without overlap are used for visualization purpose, this partitioning strategy creates the issue of inaccuracy in processing, as objects crossing boundaries of these partitions will be cut into fragments and distributed into multiple tiles as incomplete objects. These fragments would in return be recognized as separate objects by databases. Consequently, segmentation could generate inaccurate results differing from actual shapes, affecting classification or learning models that use the data. To preserve correctness, these extracted fragments have to be merged. Therefore we need to apply an additional processing step after image segmentation to consolidate shapes. Most traditional methods choose to ignore and remove incomplete objects from the result sets. While the error is relatively small for processing larger tiles (about 2% of objects at 4k x 4k image of 20X resolution), the error is not negligible when handling larger or more complex objects such as blood vessel. For large size images, the boundary effect could be even more serious due to the discard of objects. While the boundary crossing issue could be mitigated by processing tiles with larger dimension, as previously stated, such processing is inefficient. In particular, processing tiles of smaller size would achieve a better runtime thanks to parallelization and less algorithm complexity when operating on smaller tiles. In contrast, this would lead to a trade-off with a much higher percentage of boundary objects.

In our experiments, we discuss there exists an optimal tile dimension for which the runtime is minimized while preserving the accuracy of segmentation by using our spatial aggregation approach.

3 Methods

3.1 Framework Overview

In this section, we discuss the process of partitioning to facilitate distributed and parallel processing of the image in our Map Reduce based Image Analysis framework (MaReIA). We introduce our overlapping partitioning method for partitioning whole slide images into tiles with extended buffers assisting in object aggregation. For segmentation of relatively small size objects or micro-anatomic structures, such as cell nuclei, the buffer zone is

adjusted to be large enough so that objects crossing boundaries, for example, object *C* in Figure 2, will always be contained as complete objects in these buffered tiles. Consequently, as buffered tiles have overlaps, boundary-crossing objects will be identified multiple times by worker tasks processing these tiles. These duplicates of the same object will be eliminated in later steps.

However, for larger micro-anatomic objects, such as blood vessels, with no predetermined dimension bound, segmentation performed in individual tiles can generate partial fragments of the objects. Extremely large buffer would create significant redundancy in processing and thus degrade performance. Thus, for both redundancy reduction and object consolidation, we provide a spatial matching method in the later stage for identifying and removing duplicated objects efficiently. We propose the following generic approach to handle results obtained by segmentation algorithms. First, algorithms selected for corresponding classes of objects generate image masks, which are converted into geometric based representation – vectorized polygons. To handle boundary-crossing objects, we propose an efficient algorithm using R*-Tree based spatial indexing [10] to merge the geometric properties of these objects accurately and efficiently.

We illustrate the workflow on our nuclear segmentation pipeline in Figure 3. The pipeline contains three major steps: a) *tiling based on overlapping partitioning*, b) *tile based image analysis with geometry correction*, and c) *result aggregation* to eliminate the boundary effect.

In *MaReIA*, we assume whole slide images are staged onto a distributed file system. They can be stored on Hadoop Distributed File System or Amazon S3 to support data availability and fault tolerance. The first step involves extraction of rasterized images at user given resolution, their compression and save back into the distributed file system. These extraction is supported by multi-threaded processes outside MapReduce. These compressed images serve as input into the following step, image processing component.

The image processing component consists of following steps whose parallelization is fully supported by MapReduce and other distributed computing architectures: segmentation, boundary vectorization, boundary normalization, and removal of buffers. Whole object aggregation is performed to generate final results where duplicates are removed and fragments are merged.

For optimization purpose, tiling and image analysis steps are executed concurrently to maximize both CPU and I/O resource utilization, as well as to eliminate the cost of tiling in the pipeline. Note that only a fraction of total computing resource is allocated to tiling, due to its I/O intensive nature, while image analysis is much more compute-intensive.

The entire pipeline can also be logically viewed as streaming image analysis. The tiling is performed continuously generate tiles for image analysis step in configurable batches. Such setup minimizes the amount of data transferred from expensive storage locations, while maintaining only necessary tiles in local system HDFS. This is a very efficient cost-saving strategy in cloud computing environments, where incurred costs are based on total disk space usage.

3.2 Tiling with Overlapping Partitioning

In this subsection, we describe the first major step in the pipeline, along with intuition and techniques behind parallel distributed processing of tile regions. The overlapping partitioning strategy in Figure 2 generates buffered tiles combining core zone and buffer zone enlarged from basic fixed grid partitioning in Figure 4. Assuming uniform or near uniform distribution, for twodimensional whole slide images, buffered tiles are squared areas, since they minimize the total tile margin or perimeter per image given the number of tiles, which in return minimizes the expected number of boundary crossing objects.

The default width of buffer area w is determined by the upper bound on size of segmented objects, such as the maximum diameter of nuclei at given resolutions. This property guarantees that every object crossing the exterior of the core zone (e.g., illustrated by C) does not intersect with the exterior of the buffer zone and will be segmented as a complete object. Furthermore, this mechanism creates a safety margin for algorithms that have potential issues in identifying edges of objects lying on outer boundaries of the whole slide or cropped image.

However, higher w will incur higher number of objects generated multiple times by buffer zone. We configure a sufficiently small buffer width to minimize the expected number of objects to be post-processed, especially in cases of object fragments in separate tiles. In practice, the upper bound dimension of complex structures or objects might not be defined or determined, due to variability between images. Images of tissues belonging to one patient might be very different from images belonging to another. Therefore, blood vessels and relatively complex micro-anatomic structures have variable and relatively large spatial extent; the size of each fragments in a tile cannot be determined in advance without examining every image. For such case, we set the default w to be 1% of the core zone width d. The selection of such value is simple to offset for potential mismatch of boundary generation on the edge of tiles; higher buffer width allows higher sensitivity range when aggregation or matching object fragments is performed. In our cell nuclei scenario, w is set to 40 pixels, while d is set to 4096. Objects not entirely contained in the buffer area will be processed by a later algorithm to merge fragments using spatial based methods described later.

The tiling component in *MaReIA* is fully parallelized. As aforementioned, whole slide images designed by commercial vendors are often stored in a hierarchical structure when obtained from digital scanners. For images staged onto a distributed file system (HDFS or Amazon S3) with a large block based storage and higher latency, direct random access is not possible.

We provide a distributed multi-threading based approach to tile generation with compression support. A tiling supervisor process invokes worker tasks on each node for tiling purpose. Each image is fetched onto the local file system on physical node, where random access is possible to extract individual regions or tiles. Each process corresponding to a single image starts a number of threads to extract data using random access, compress the data and finally write the content back to the distributed file system for image analysis. The effective number of threads is only restricted by the physical I/O capacity. We can achieve higher horizontal

scalability by invoking tiling processes on multiple nodes, similar to shared nothing MapReduce parallelization.

Furthermore, to reduce network load, we compress every tile separately. Since each uncompressed tiles is about 64 MB, most common compression algorithms, in our case ZLIB, provide relatively good compression rate, resulting in 20 MB average per compressed data tile. Each thread uses random access I/O and small fraction of CPU time to process its assigned tiles. The output from tiling individual images is the set of tiles compressed in ZLIB format with the file name formed by concatenation of the image ID and its top-left coordinates, and forms a unique key for MapReduce or key based processing, e.g., *TCGA-27–1836-01Z-DX20000004096–0000016384*, where 4096 and 16384 are reference coordinates used for spatial support in later steps. These identifiers are used to manage object-level information and aggregation support.

3.3 Tile Based Image Analysis with MapReduce: Map

The image analysis component consists of two major steps: 1) tile based object segmentation to generate vectorized or spatial geometry of objects; and 2) aggregation of tile based results by merge and elimination (Figure 3). The two phases match perfectly into the map and reduce phases of MapReduce. The tile based segmentation phase consists of following procedures: a) Object segmentation on image arrays producing image masks; b) boundary vectorization based on union find algorithm to convert contours of nuclei from mask images into shapes which might not be valid polygons; c) boundary normalization into valid polygons and multipolygons; and d) early elimination of objects in the buffer zone which do not intersect core zones. The last step d) is optional and not used when segmenting blood vessels or complex structures.

Object segmentation.—Segmentation refers to the process of generating image mask from raw image raster data. Each tile can be processed with one or more segmentation algorithms depending on number of object types users require. Objects are identified by their assigned object class attributes that are passed along their geometry description in the pipeline. In addition, same sets of objects under different parameter sets can be generated in a single pass over the data. This is common when performing sensitivity analysis over image segmentation algorithms; for instance, users can vary threshold and filter size in watershed transformation when segmenting nuclei. This makes the framework highly extendable in other applications. The framework interface only requires the algorithms to produce black and white mask images, where segmented objects are white filled (F3 in Figure 3).

Boundary vectorization.—To accurately support further analysis, segmented nuclei need to be converted into vector based representation. Contours of nuclei in the mask images are extracted as geometric shapes representing boundaries. The vectorization process translates local coordinates within a tile into the absolute global coordinates. The global coordinates are adjusted based on reference coordinates obtained from the MapReduce input key during tiling. The global coordinate system is required so objects can be uniquely identified and processed in the later *Result Aggregation* step.

Boundary normalization.—Though boundaries were generated by algorithm, frequently they contain invalid geometric shapes, which may not be valid polygons [6]. For example, the shape F5 is not be closed topologically due to variability and randomness in contour traversal in boundary vectorization. These invalid polygons cannot be used in spatial or geometric operations in later analysis. They have to be corrected or adjusted as illustrated in Figure 6.

We used our previously developed boundary correction tool to convert invalid boundaries into valid polygons or geometries. The tool gracefully handles the following problems: open-ended contour, twisted contours, collinear points, duplicate points, dangling lines, multipolygons, self-intersection, and sharp pointed vertices. The tool consists of an efficient pipeline of components developed on top of existing libraries, including Boost [1] and Clipper [5].

Isolated buffer polygon removal.—For object geometries entirely contained in the buffer zone but not in the core zone, for instance, F8 in Figure 3, we eliminate them early, as they are not boundary crossing objects, they will be excluded from post-processing. By the setup of partitioning, exactly one copy of the object will be generated in another tile. This reduces the number of objects involved in aggregation, thus improves performance. The check is simply conducted by performing intersection between Minimum Bounding Rectangles (MBRs) of our objects with the buffer zone represented, the buffered objects or polygons representing nuclei can be removed.

MapReduce based implementation.—In the context of MapReduce, the tiles generated from partitioning form a natural unit for MapReduce parallel based processing. The tile based image processing is implemented as a Map function, where the input is a key-value pair of *(tile-id, tile-data)*. MapReduce will scan the input collection of all tiled images, and each tile image becomes a unit to process by the Map function, which is executed in parallel as MapReduce tasks running on a cluster. The output of Map function contains records of *(image-id—label, polygon,attributes)*, where the *image-id* encodes the image ID and the top-left coordinates of the tile. Attributes include object class or type, depending on whether multiple segmentation algorithms have been used.

To distinguish geometries intersecting with the exterior boundary of the core zone, an additional attribute is appended to the key *tile-id* to label it, denoting these polygons as a special class. Only these specially labeled objects will be kept for the following result aggregation. Objects without label are directly output to their final locations and stored in databases.

3.4 Result Aggregation with MapReduce: Reduce

The tile based results will be aggregated and merged in the Reduce phase of the MapReduce job, and the composite key is image id and object class. There is a shuffling step between Map and Reduce phases, which groups polygons from the images based on image IDs and their respective object class. This guarantees that objects of the same type generated from the same image will be processed by the same reduce task.

There are two primary types of geometries: a) nonoverlapping geometry, e.g., polygons in F9 in Figure 3; and b) overlapping geometries – labeled through appending a tag in tile-id, e.g., polygons in F10. These overlapping objects have duplicates and will be matched and merged to consolidate results.

MaReIA uses a spatial index based algorithm to consolidate polygons that are fragments of the same object. An R*-Tree commonly used in spatial domain is first built based on MBRs of all the overlapping objects of the same image [10]. The R*-Tree nodes have containment relationship between MBRs in a parent node and MBRs in a child node. The R*-Tree size is very small due to its efficiency and also number of overlapping polygons, which are only small percentages of the total number of objects in each image. The indexing tree provides logarithmic access time and serves as a fast filter tool in removing duplicates. Incoming boundarycrossing polygons are placed in a queue for index based identification, and each first polygon of duplicates will be output in the output queue.

Next, we describe the importance and algorithmic approach in our aggregation. For complex micro-anatomic structure, merging objects are essential, since geometries generated from a single tile might represent only a partial object segmented by the buffer boundary, such as in Figure 7. The underlying principle is using *spatial intersection* on geometries to determine whether partial objects belong to the same object, and performing *spatial union* to generate the final complete objects.

For efficient object consolidation processing, we use a modified version of our Hadoop-GIS, a scalable spatial data framework supporting spatial and geometric operations.

The consolidation algorithm we use to merge objects is based on the breadth-first search algorithm (See 1). The algorithm heuristically minimizes the number of spatial access and geometric operations to join fragments of the same object. The threshold parameter η is used to separate objects touching or slightly overlapping from fragments belonging to the same object in the pipeline. The algorithm has a time complexity of $O(n \log n)$, where n is the total number of boundary crossing objects in an image. In practice, the percentage of boundary-overlapping objects is small; thus the additional processing cost is rather insignificant thanks to the usage of R*-tree in indexing.

The complexity of the algorithm is O(n log n), where n is the number of boundary objects, and also proportionally can be viewed as the total number of objects in an image. The approach resembles a spatial self-join query on the spatial data sets.

MapReduce based result aggregation.—In the MapReduce context, each reduce tasks sequentially process cleaned polygons of one or multiple images. MapReduce mechanisms allow reduce tasks to perform object deduplication and object consolidation in a single pass over the data. In other words, non-boundary polygons are directly written to output, while boundary polygons are retained in memory and passed to the object consolidation algorithm.

3.5 Parameter Optimization

Parameter optimization is critical to achieve good performance. In our case it is determining a parameter set suitable to streamline processing. The parameters in discussion are MapReduce parameters, tiling, compression, and aggregation parameters. For each step in the pipeline handled by MapReduce, MapReduce parameters refer to the amount of resources (memory and CPU) for each individual processing task in MapReduce environment. Tiling parameters include most notably the tile dimension. Additional compression parameters include the encoding type and compression ratio, while aggregation furthermore includes thresholds for joining and aggregation.

We observed that tiling parameters, in particular tile dimension, affects runtime most significantly. Aggregation thresholds in contrast affects the accuracy and correctness of overall segmented objects.

In practice, the computational bottleneck in the image processing component is the segmentation and vectorization in the Map step. A pivotal parameter is for instance the tile dimension. We observed that lowering tile dimension could increase the pipeline throughput. Particularly, segmentation algorithms with runtime complexity of higher than $O(nlogn)$ benefit from lower tile dimension. For instance, our nuclei segmentation has a runtime complexity of $O(n^2)$, where n is the tile dimension (edge size). One of blood vessel algorithms has a complexity of $O(n^3)$ and thus benefits more from smaller tile dimension. However, the trade-offs are the larger amount of rasterized data being transferred due to overlapping buffers and lower tile compression ratio, effectively increasing data transfer across the network. Furthermore, the number of boundary-crossing objects will increase, resulting in increasing cost of result aggregation step. It can be shown that the number of boundary-crossing objects is linear to the number of tiles, which is inversely proportional to the square of tile dimensions.

Here we illustrate our derivation of the optimal tile dimension with an example for nuclei segmentation. The following represents our notations.

S	Total number of pixels per image
d	Core zone width (tiling dimension)
n	Number of tiles in each image
C_{tile}	Cost of tiling
C_{map}	Cost of Map step
$C_{shuffle}$	Cost of MapReduce shuffling sorting
C_{reduce}	Cost of Reduce/aggregation
α, β, γ	Map cost coefficients
η	Boundary object increase coefficient
ω	Aggregation cost coefficient
T	Base number of boundary objects

Denote $C(d)$ as the total cost function of core zone width. As aforementioned, the total processing cost is the sum of tiling and MapReduce processing. MapReduce consists of three phases: Map, sorting and shuffling and Reduce. In other words,

$$C(d) = C_{tile} + C_{map} + C_{shuffle} + C_{reduce} \quad (1)$$

Thanks to MapReduce scheduler and distributed engines, minimizing the total cost will result in minimizing the overall runtime. Thus, an optimal d_0 occurs when $d_0 = argmin(C(d))$.

To simplify our analysis, we exclude the tiling cost, since the throughput of tiling is adjusted to match the throughput of image analysis and aggregation. We also show in Sec. 4 that tiling cost does not change significantly when tile dimension is in the range of from 500 pixels and up.

For the map phase including image analysis, vectorization and normalization, the overall complexity $C_{map} = O(d^2)$ because of the dominating complexity of our nucleus segmentation algorithm. Thus the cost for each tile can be written as $C_{map}(d) = \alpha d^2 + \beta d + \gamma$, where the cost coefficients can be estimated by experimental runtime; in our experiment we test d at 500, 1k, 2k, 5k and 10k pixels to derive α, β, γ. For segmentation algorithms with higher complexity, there are more coefficients to represent the higher order factors. Since there are in total S pixels in an image, there would be $n = \frac{S}{d^2}$ tiles or Map tasks to be run. The total cost of Map step would be the product of each tile cost and the number of tiles.

Next, in estimating the cost for sorting and shuffling boundary objects and aggregation (Reduce), spatial indexing with R-tree is used, the cost of shuffling each object fragment is $O(log(t))$, where t is the number of objects in the aggregation pipeline. Therefore, $C_{shuffle} + C_{reduce} \approx \omega T log(T)$. On the other hand, since the increase of of boundary crossing objects that need to be shuffled is approximately linear to the number of tiles $\approx n = \frac{1}{d^2}$, the total number of objects processed in shuffling and Reduce step is $t = T\left(1 + \frac{\eta}{d^2}\right)$.

Substituting above expressions into Eq. 1, we obtain:

$$C(d) \approx \frac{S}{d^2}\left(\alpha d^2 + \beta d + \gamma\right) + \omega T\left(1 + \frac{\eta}{d^2}\right) log\left(T\left(1 + \frac{\eta}{d^2}\right)\right) \quad (2)$$

By solving $\frac{\partial C}{\partial d} = 0$, we obtain a critical point that could be the local minimum for runtime. The values of the coefficients such as *alpha* can be obtained by extrapolation and profiling when performing analysis on a single image; these coefficients are invariant to the size of the datasets. Due to space constraint, we found that for nucleus segmentation, $d_0 \approx 2500$. In Sec.

4, we experimentally found that the optimal tile dimension is between 2800 and 3200. This could be attributed to the relaxed assumptions we previously discussed.

Overall, in this case of whole slide images on commodity clusters, the overall dominating bottleneck is CPU computation and the memory requirement is often dictated by the segmentation algorithm. To take 100% advantage of CPU, we determine the maximum available amount of available memory and number of processors, which in turns determines the maximum amount of memory, specifically heap space available to segmentation algorithms.

Consequently, the maximum amount of memory helps selecting the maximum dimension of a tile that can be processed. In practice, we use this upper-bound as a starting value for the tile dimension.

For whole slide images, objects are often uniformly distributed in the image; thus we could determine an approximate tile dimension minimizing performance run time based on system and image parameters.

4 Performance and Evaluation

In this section, we describe experiments used to evaluate the framework and discuss their implications.

Setup.

We use a data set of 475 whole slide images of brain tumor from TCGA portal [1], with size of 1.3 TB. The data set contains more than 180 million nuclei to be extracted. We use Amazon EMR cluster consisting of up to 100 *c3.xlarge* nodes, each equipped with 7.5GB of RAM, two 40GB solid state drives, and a Intel Xeon E52680 2.8 GHz processor with four cores. The maximal number of processing cores is 400.

Execution Time Profiling of the Pipeline.

First, we analyze the computational complexity of the framework if steps in the framework were executed sequentially. We show the time breakdown of each component for processing images with core tile dimension of 4k x 4k using 40 processing cores in Fig. 8. We also break down the execution time of the image processing pipeline. Data staging takes 8.7%, tiling takes 13.1%, segmentation takes 57.7%, vectorization and normalization takes 10%, and aggregation takes 10.5%. The merging of duplicated polygons in the aggregation step is only a small fraction of the total cost due to the effective spatial indexing based approach. With the approach of pipelining tiling during image processing step, we reduce the tiling step relative cost to less than 1% of the pipeline, accounting for the generation of tiles for the first batch of images.

[1] https://tcga-data.nci.nih.gov/

Tile Generation Performance.

The whole slide images are stored on Amazon S3. The tiling is performed in parallel on smaller sub clusters, and the generated tiles are copied from local disk back to Hadoop Distributed File System shared by the larger cluster. We only use 10 nodes to perform tiling as they are sufficient to generate tiles quickly for the remaining part of the pipeline. We find that multi-threading on each node will increase the tiling speed significantly due to better utilization of CPU and I/O. As shown in Figure 9, the total tiling time for all images drops from over 160 minutes to less than 12 minutes when the number of threads per node is increased from 1 to 16. This is attributed to the fact that I/O is the bottleneck or primary constrained resource in tiling. Therefore, for better tiling performance, solid state drives (SSDs) could deliver high performance reading with multi-thread reading [14]; with faster disks the number of tiling nodes can be reduced, allowing more resource or nodes to be performing the image analysis step.

Overall Scalability of MapReduce Based Image Processing.

Figure 10 shows the performance of MapReduce based image processing, with increasing number of processing cores, which demonstrates an almost linear scalability for nuclei segmentation. Note that the runtime is in log-scale. By increasing the number of cores from 32 to 400 (a factor of 12.5), the time drops from 1045 minutes to 90 minutes (a factor of 11.6). The projected time for processing without any parallelization is approximately 23 days. Faster processing can be easily achieved through increasing the number of processing cores. We can see MaReIA scales well with growing data sets, achieving also close to linear scalability with respect to dataset sizes.

Sensitivity Analysis of Tile Dimension.

Next we analyze the effect of adjusting the tile dimension parameter on individual component, assuming unchanged hardware configuration. As aforementioned, Figure 9 shows tiling time when the tile dimensions are approximately 4k x 4k. Fig. 11 shows tiling performance of 50 images using 10 threads for varying dimensions of tile edge width from 512 to 8k pixels. We observe the major factor in tiling time with different dimensions is compression time. Our data shows that tile width smaller than 500 pixels incur a relatively high compression cost.

In contrast, tile dimensions smaller than 1024 pixels show I/O performance degradation. First, smaller tiles incur a large total amount of data due to the use of buffer. Second, compressing and writing a larger tile is more I/O efficient than operating on multiple small tiles sequentially.

Thus, when including the runtime of the image analysis component, the total runtime shows a potential global minimum of runtime or the sweet spot to achieve the fastest processing as shown in Fig. 12. The figure shows that performance of the case in which 10 nodes with 8 threads per node were used for tiling and 200 processing cores were used for the image analysis component. We investigate several factors that have significant effects on overall runtime. First, most segmentation algorithms are of order higher than linear complexity with respect to tile dimension, implying that smaller tile dimension is favored. The segmentation

algorithm used in the framework has a quadratic complexity on average; thus it appears to be more desirable to use smaller tile size. The processing in each tile benefits from a better degree of parallelization. The more complex the algorithm, the smaller the tile dimension should be. On the other hand, smaller tiles create approximately linearly proportionally more boundary objects, resulting in a more expensive boundary object aggregation cost.

Second, smaller tile dimension could generate sparse or empty tiles which create an overhead cost to MapReduce. Third, the overhead cost of creating more tasks to handle individual tiles become significant for smaller tile size, since the effective processing time on each tile of each task is amortized contributing to worse performance. Thus, the best tile dimension results from the combination of the aforementioned factors in both tiling and image analysis. In our experiments, the near optimal tile dimension is approximately in the range of 2800 to 3200 pixels. The actual precise optimal tile size might vary between classes of segmented objects, images and data sets.

Cost Estimation.

We evaluated the cost for processing images on Amazon cloud. Amazon EMR has different node types with different prices. For example, *c3.xlarge* comes with 4 cores and costs $0.053 per hour if being used for Elastic MapReduce, and $0.210 per hour if being used for EC2. We only use 10 nodes for tiling as it is sufficient to produce tiles quickly, and 100 nodes with 400 cores for MapReduce based image processing on EMR. The total cost is approximately $12.7, including $10.6 for image processing and $2.1 for tiling. Amazon S3 storage costs approximately $0.030 per GB per month. As all the uploading, processing and downloading could be finished within 2 hours, it will take less than two hours to store the images and tiles. The storage cost is about 20 cents and can be ignored. The data uploading is free, and data downloading costs $0.12 per GB. The result is about 100GB, which costs $12 to download. Thus, the total cost will be about $25, with an average cost of about 6 cents per image.

5 Conclusions

Distributed and parallel, in particular MapReduce based, programming model and cloud computing have become a de facto standard for large scale data analytics due to the scalability, fault tolerance, simplicity and costeffective operations. In our application of whole slide imaging analysis, we looked at feasibility of applying the software stack for large scale whole slide image analysis, by solving the critical issues of partitioning, inaccuracy, and parallelization. We propose a generic distributed workflow with tile level parallelism, by developing a spatially buffered partitioning method in addition to a novel indexing based matching method to alleviate boundary effect. Objects in the pipeline are converted into vector representations in a global coordinate system specific to each whole slide image, facilitating a robust and accurate processing. The aggregation eliminates duplicate and merge fragments based on spatial join and spatial union operations efficiently with spatial indexing support. Parallelization is achieved by MapReduce, resulting in almost linear scalability with respect to the amount of input data. The approach is very robust and can incorporate any tile based image segmentation algorithm. We showed based on our

Amazon EMR benchmark tests that *MaReIA* is very costeffective and provides high throughput to users.

6 Acknowledgements

This research is supported in part by grants from National Science Foundation ACI 1443054 and IIS 1350885, National Institute of Health K25CA181503, and CNPq.

References

1. boost c++ libraries. http://www.boost.org/, 2003.
2. Apache hadoop. http://hadoop.apache.org.
3. Apache spark. http://spark.apache.org.
4. Apache spark. http://storm.apache.org.
5. Clipper library. http://www.angusj.com/delphi/clipper.php.
6. Geospatial standard. http://www.opengeospatial.org/standards/sfs.
7. Abadi M, Barham P, Chen J, Chen Z, Davis A,Dean J, Devin M, Ghemawat S, Irving G, Isard M, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.
8. Aji A, Wang F, and Saltz JH. Towards Building AHigh Performance Spatial Query System for Large Scale Medical Imaging Data. In *SIGSPATIAL/GIS*, pages 309–318. ACM, 2012. [PubMed: 24501719]
9. Aji A, Wang F, Vo H, Lee R, Liu Q, Zhang X, andSaltz J. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. Proc. VLDB Endow, 6(11):1009–1020, Aug. 2013.
10. Beckmann N, Kriegel H, Schneider R, and Seeger B.The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.
11. Cooper LA, Kong J, Gutman DA, Dunn WD,Nalisnik M, and Brat DJ. Novel genotype-phenotype associations in human cancers enabled by advanced molecular platforms and computational analysis of whole slide images. Laboratory investigation, 95(4):366–376, 2015. [PubMed: 25599536]
12. Cooper LAD, Kong J, Gutman DA, Wang F, Gao J, Appin C, Cholleti S, Pan T, Sharma A, Scarpace L, Mikkelsen T, Kurc T, Moreno CS, Brat DJ, and Saltz JH. Integrated morphologic analysis for the identification and characterization of disease subtypes. J Am Med Inform Assoc, Jan. 2012.
13. Dean J and Ghemawat S. Mapreduce: Simplified dataprocessing on large clusters. Commun *ACM*, 51(1):107–113, 2008.
14. Wang XZF, Lee R and Saltz J. Towards buildinghigh performance medical image management system for clinical trials. In *SPIE Medical Imaging*, pages 762805–11, 2011.
15. Foran DJ, Yang L, Chen W, Hu J, Goodell LA, Reiss M, Wang F, Kurç TM, Pan T, Sharma A, and Saltz JH. Imageminer: a software system for comparative analysis of tissue microarrays using content-based image retrieval, high-performance computing, and grid technology. JAMIA, 18(4): 403–415, 2011. [PubMed: 21606133]
16. Gu L and Li H. Memory or time: Performance evaluation for iterative operation on hadoop and spark. In High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC EUC), 2013 IEEE 10th International Conference on, pages 721–727. IEEE, 2013.
17. Hare JS, Samangooei S, and Dupplaw DP. Openimaj and imageterrier: Java libraries and tools for scalable multimedia analysis and indexing of images In Proceedings of the 19th ACM international conference on Multimedia, pages 691–694. ACM, 2011.
18. Kong LCJ, Moreno C, Wang F, Kurc T, Saltz J,and Brat D. In silico analysis of nuclei in glioblastoma using large-scale microscopy images improves prediction of treatment response. In *EMBC*, 2011.

19. Kong J, Cooper LAD, Wang F, Gao J, Teodoro G,Scarpace L, Mikkelsen T, Schniederjan MJ, Moreno CS, Saltz JH, and Brat DJ. Machine-based morphologic analysis of glioblastoma using whole-slide pathology images uncovers clinically relevant molecular correlates. PLoS One, 8(11), 11 2013.

20. Kothari S, Phan JH, Stokes TH, and Wang MD.Pathology imaging informatics for quantitative analysis of whole-slide images. Journal of the American Medical Informatics Association, 20(6): 1099–1108, 2013. [PubMed: 23959844]

21. Liang Y, Wang F, Treanor D, Magee D, Roberts N,Teodoro G, Zhu Y, and Kong J. A framework for 3d vessel analysis using whole slide images of liver tissue sections. International journal of computational biology and drug design, 9(1–2):102–119, 2016. [PubMed: 27034719]

22. Markonis D, Schaer R, Eggel I, Muller H, and Depeursinge A. Using mapreduce for large-scale medical image analysis. arXiv preprint arXiv:1510.06937, 2015.

23. Sweeney C, Liu L, Arietta S, and Lawrence J. Hipi: a hadoop image processing interface for image-based mapreduce tasks. Chris. University of Virginia, 2011.

24. Teodoro G, Pan T, Kurc T, Kong J, Cooper L, Podhorszki N, Klasky S, and Saltz J. High-throughput analysis of large microscopy image datasets on cpu-gpu cluster platforms. In *IPDPS*, pages 103–114, 5 2013.

25. Vo H, Kong J, Teng D, Liang Y, Aji A, Teodoro G,and Wang F. Cloud-based whole slide image analysis using mapreduce In VLDB Workshop on Data Management and Analytics for Medicine and Healthcare, pages 62–77. Springer, 2016.

26. Wang F, Oh TW, Vergara-Niedermayr C, Kurc T,and Saltz J. Managing and querying whole slide images. In *SPIE Medical Imaging*, 2012.

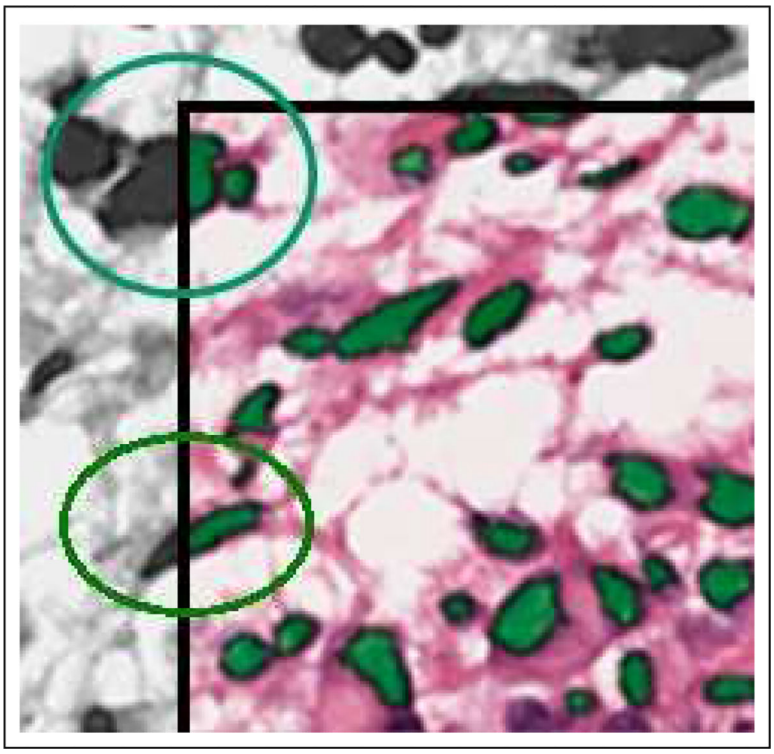

Fig. 1.
Examples of objects crossing grid boundary

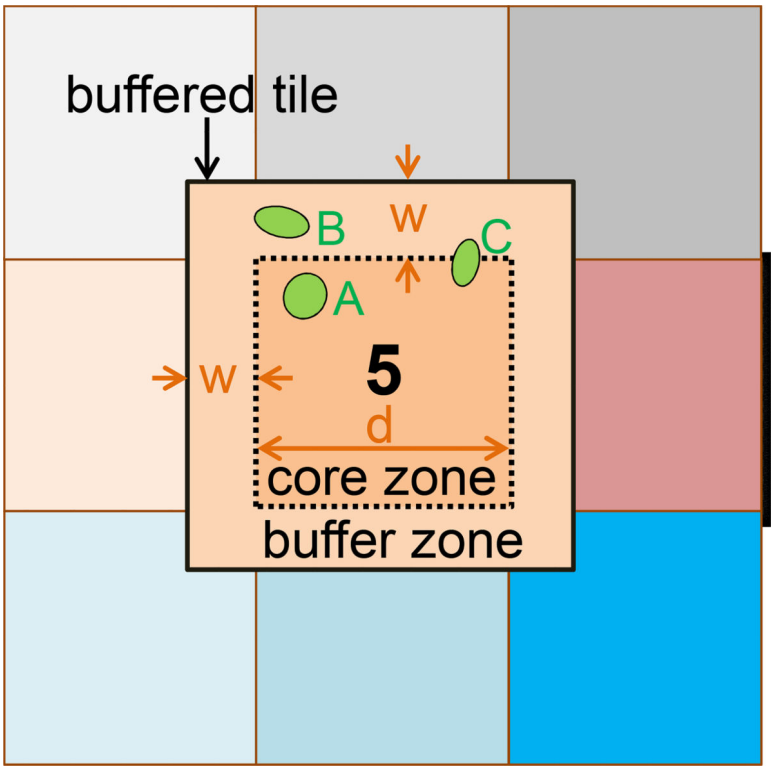

Fig. 2.
Overlapping partitioning

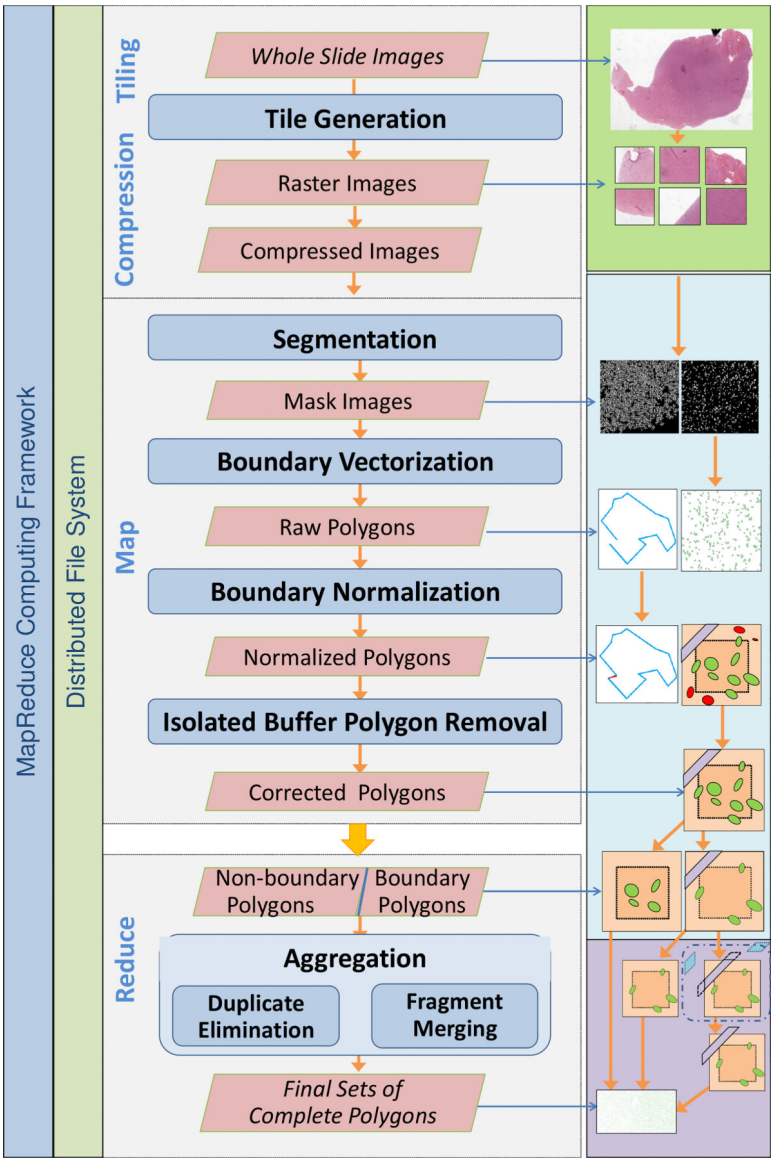

Fig. 3.
Overview of the system workflow

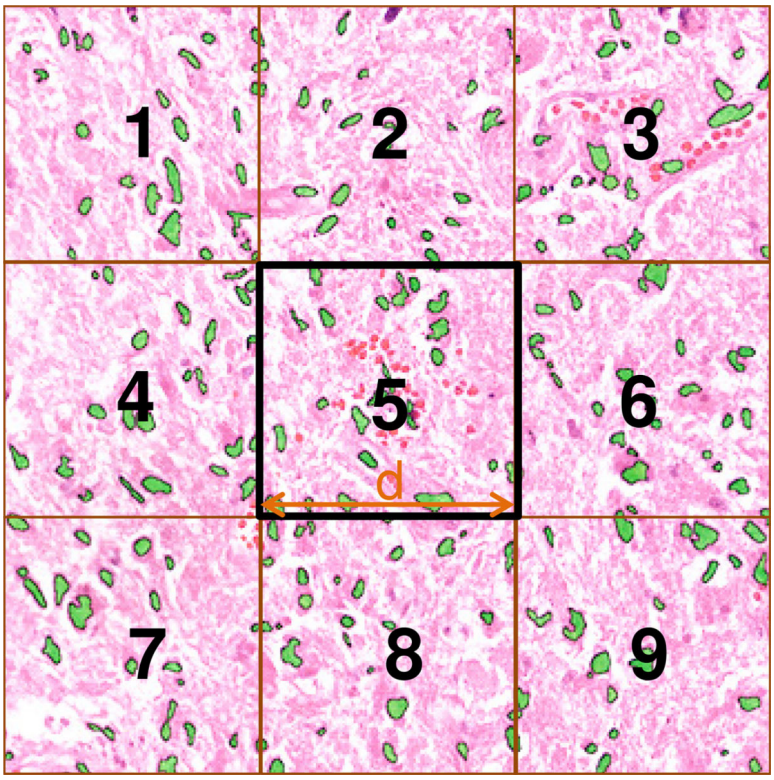

Fig. 4.
Fixed grid partitioning

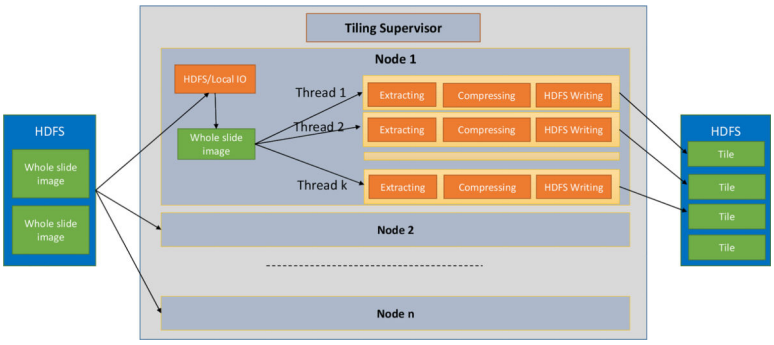

Fig. 5.
Distributed multi-threading tiling component

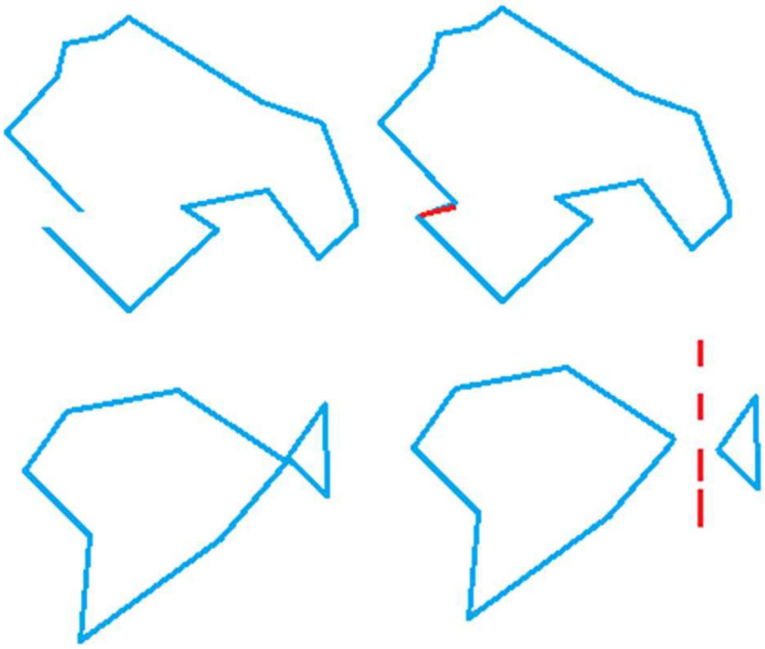

Fig. 6.
Examples of invalid and normalized valid objects

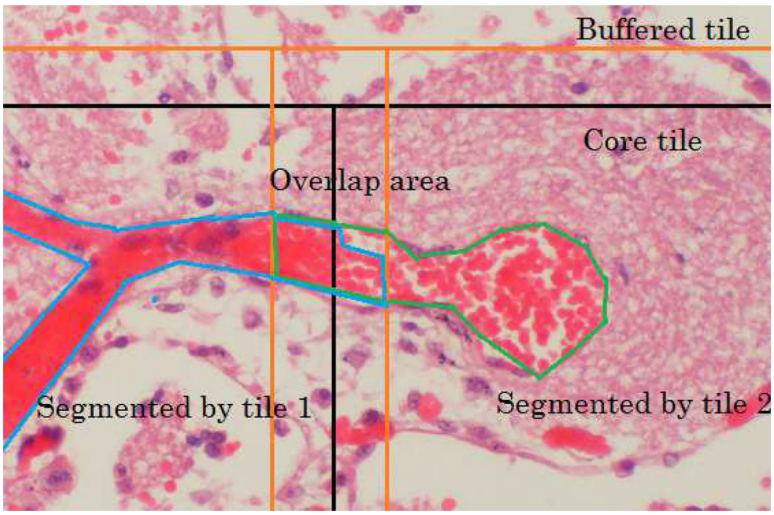

Fig. 7.

Object merge example from partial fragments (blue and green). Core tile boundary is black; buffer boundary is orange.

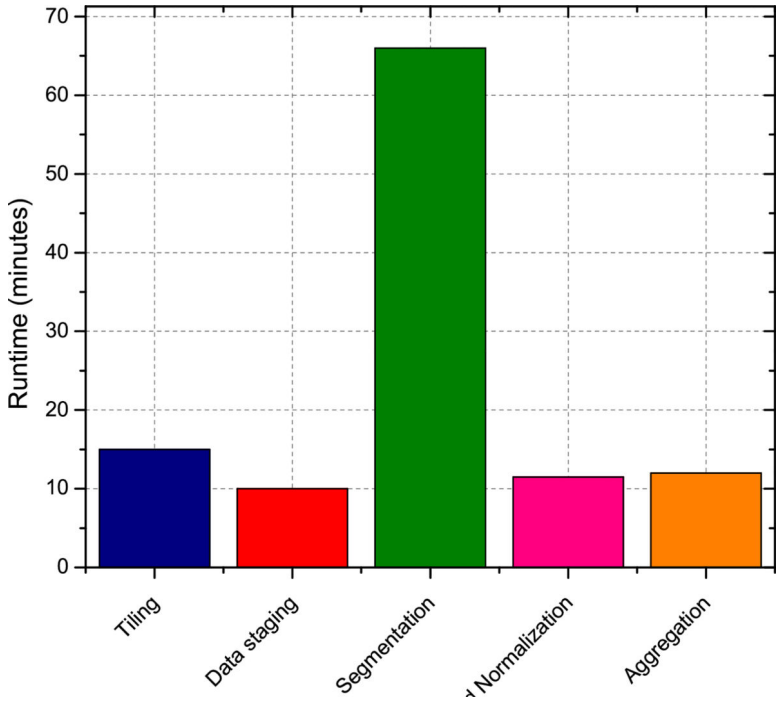

Fig. 8.
Sequential pipeline execution breakdown

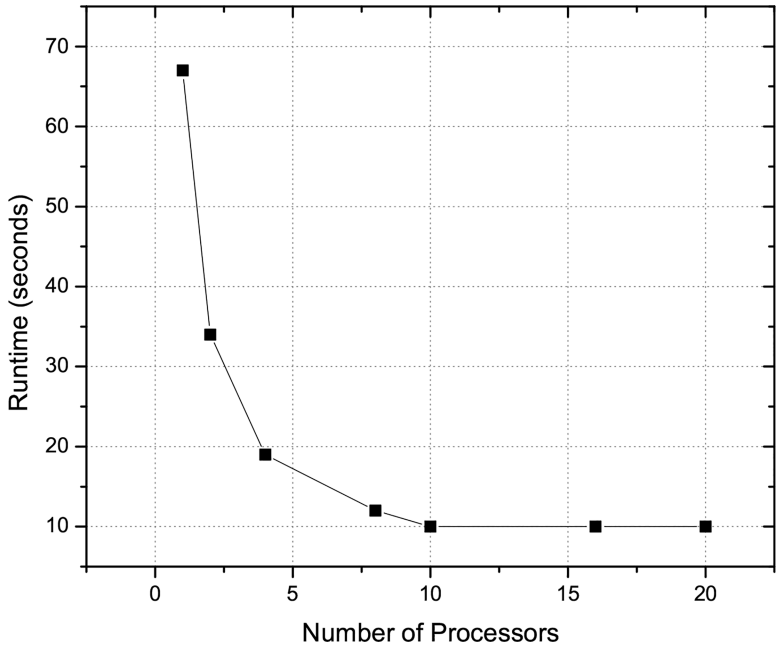

Fig. 9.
Performance of multi-thread tiling on 10 nodes

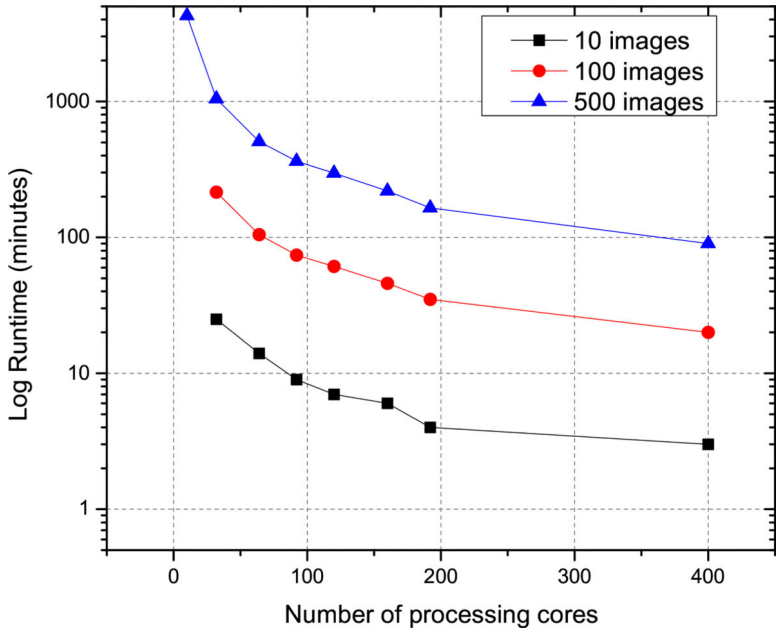

Fig. 10.
Scalability of MaReIA

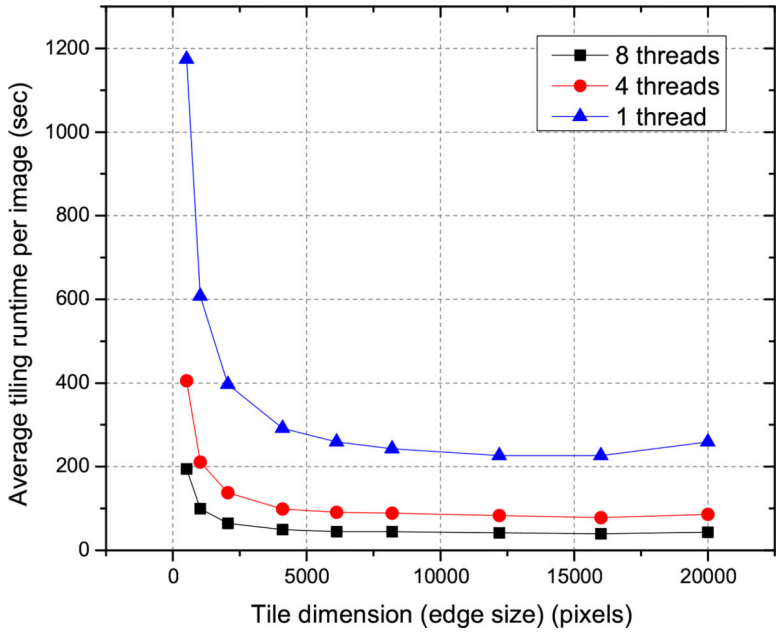

Fig. 11.
Tiling performance for different tile dimension

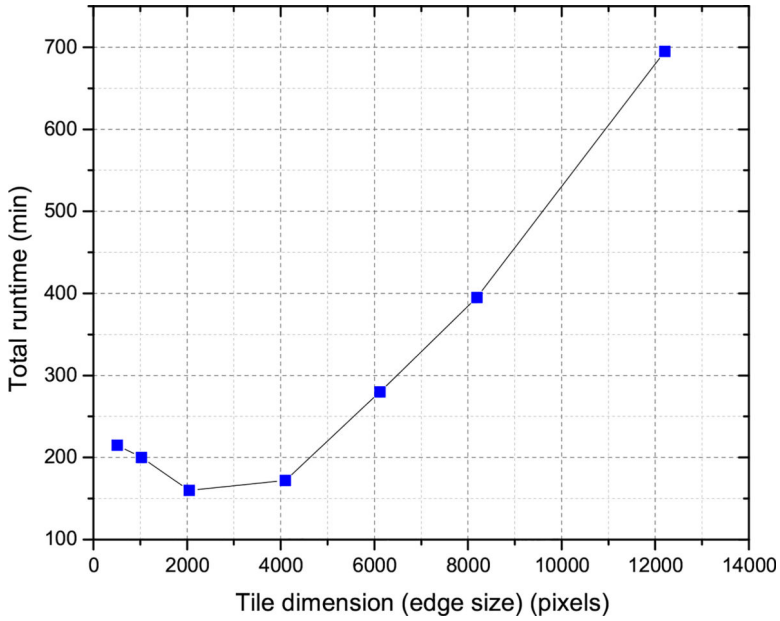

Fig. 12.
Total framework runtime in seconds for varying tile dimensions

Algorithm 1:

Object Consolidation

```
1  while there are boundary objects to read do
2      read object o;
3      vectorObjects.push(o);
4  end
5  constructR*tree(vectorObjects);
6  for object o in vectorObjects do
7      if o in queue continue;
8      else for object t in vectorObjects do
9          if s not in queue2 and
           s.overlapWithThreshold(η) then
10             queue2.push(t);
11             queue.remove(t);
12         end
13     end
14     obj = queue.head();
15     for object u in queue2 do
16         obj = obj.join(u);
17     end
18 end
```