Spatial Computations over Terabyte-Sized Images on Hadoop Platforms

Peter Bajcsy, Phuong Nguyen, Antoine Vandecreme, Mary Brady
Software and Systems Division, Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD
e-mail: {peter.bajcsy, phuong.nguyen, antoine.vandecreme, mary.brady}@nist.gov

***Abstract*—Our objective is to lower the barrier of executing spatial image computations in a computer cluster/cloud environment instead of in a desktop/laptop computing environment. We research two related problems encountered during an execution of spatial computations over terabyte-sized images using Apache Hadoop running on distributed computing resources. The two problems address (a) detection of spatial computations and their parameter estimation from a library of image processing functions, and (b) partitioning of image data for spatial image computations on Hadoop cluster/cloud computing platforms in order to minimize network data transfer. The first problem is solved by designing an iterative estimation methodology. The second problem is formulated as an optimization over three partitioning schemas (physical, logical without overlap and logical with overlap), and evaluated over several system configuration parameters. Our experimental results for the two problems demonstrate 100% accuracy in detecting spatial computations in the Java Advanced Imaging and ImageJ libraries, a speed-up of 5.36 between the default Hadoop physical partitioning and developed logical image partitioning with overlap, and 3.14 times faster execution of logical partitioning with overlap than the one without overlap. The novelty of our work is in designing an extension to Apache Hadoop to run a class of spatial image processing operations efficiently on a distributed computing resource.**

Keywords: Spatial image operations; Hadoop; Image partition; Distributed computing

I. INTRODUCTION

Our overarching goal is to automate transitions of image processing computations from a single desktop computer to a cloud/cluster computational resource. Among all possible image processing operations, we identified a class of *spatial* image computations that are suitable for such a transition. *The spatial image computations operate on a set of contiguous image regions that can be performed in parallel*. The regions in a set might vary in size and shape, and might spatially overlap. The sub-division of spatial image computations is summarized in Table 1. The rows and columns of Table 1 represent characteristics in terms of image region size, presence of overlapping image regions to compute the resulting values, desired image partitions to co-locate image data with the computation on a cluster node, and the inputs for performing desired image partitions. The term "logical" refers to a partition based on an image pixel location while "physical" denotes a partition based on a file storage location. If a spatial computation and image values are not co-located then overall computational time increases due to network transfers of data to each computational node. In this *paper, one of our technical goals is to design an image sub-area distribution schema that minimizes the network transfer for spatial computations*.

TABLE 1: SUB-DIVISION OF SPATIAL IMAGE COMPUTATIONS AND ITS RELEVANCE TO IMAGE PARTITION.

Types of spatial computations: examples	**Input Image Region**	**Overlap type**	**Desired Image Partition**	**Input to Logical Partition**
Pixel-based: Thresholding	Fixed size	No overlap	Physical or logical without overlap	None
Kernel-based: Convolution	Fixed size	With overlap	Logical with overlap	Kernel area size
Segment-based: Feature extraction	Variable size	No overlap	Logical without overlap	Mask
Bounding box-based: Background correction	Variable size	With overlap	Logical with overlap	Bounding boxes

The motivation for our work comes from the fact that implementations of image spatial computations are ubiquitous in image software packages, embedded in many image processing methods, designed to run on desktops, and applied by a large number of users in imaging and image processing communities. However, the current desktop implementations do not perform successfully on terabyte-sized images (i.e., reporting out-of-memory error or their computational time is prohibitive), and do not parallelize computations efficiently in cluster/cloud computing environments. Based on Table 1, execution time of spatial computations on a computer cluster can be minimized by using image partitioning schemas and design patterns for parallel software [1] that consider image regions (logical partitions) rather than file arrays on a disk (physical partitions). Once image regions are distributed to multiple computational nodes, processing can be launched in parallel without additional network transfers of image data. One of the popular open source middleware platforms for parallel

execution is Apache Hadoop [2], [3]. It manages the physical data partitioning, distribution across cluster nodes, job management and result aggregation. Although Hadoop has been widely used by researchers and industries for parallel execution on distributed computational resources, and it is known for its relative programming simplicity [3]–[5], it lacks support for image processing. Specifically, Hadoop does not take advantage of logical partitions of images and therefore does not deliver the possible execution speed-up for spatial image computations.

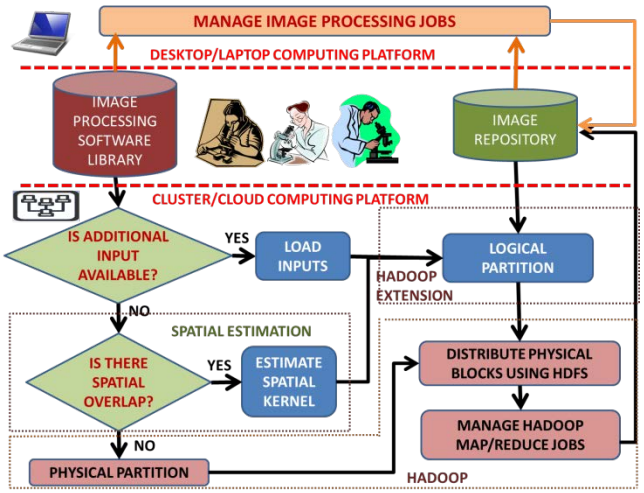

Fig. 1. Execution options for scientists to run spatial image computations on desktop/laptop or cluster/cloud computing platforms. HDFS denotes the Hadoop Distribution File System.

To address the aforementioned deficiencies of executing spatial image computations on Hadoop-managed distributed computational resources, we investigated two problems: (1) detection of spatial image computations in a library of image processing methods, and parameter estimation of detected spatial computations, and (2) distribution of image regions according to logical partitions to be processed by a single cluster node. The two problems are illustrated in Fig. **1** from the perspective of a bench scientist who is in a transition from a desktop to cluster platforms. In order to handle terabyte-sized images on cluster/cloud platforms, new software has to be designed to decide whether physical or logical image partitioning is needed, estimate parameters of logical partitions and then distribute images accordingly to take a full advantage of distributed cluster/cloud platforms.

The design of algorithms to detect and estimate spatial overlap is approached by a spatial transformation test. The spatial transformation test is performed by comparing the outcomes of two image processing sequences (image crop → software functionality) and (software functionality → image crop). The design of a logical image partitioning schema is an extension to Hadoop middleware with MapReduce implementations of image processing functionalities. The image regions (logical partitions) are packaged into physical blocks and distributed to the Hadoop Distribution File System (HDFS) such that each cluster node would have all input image pixels needed to derive output image values.

The impact of our work is in providing scientists with a methodology for detecting and distributing spatial image computations using logical image partitions. The novelty of our work is in designing the detection methodology for spatial image computations and evaluating experimentally multiple physical and logical partitioning schemas on a Hadoop computer cluster. We have evaluated the developed solutions using Java Advanced Imaging [6] and ImageJ [7] libraries.

This paper is organized as follows. Section II outlines related work. In Section III, we present the theoretical underpinnings of the approaches to (a) the spatial image computation detection and parameter estimation problem and (b) the design of image partitioning schemas for spatial image computations. Section IV describes the experimental results including hardware and test data sets, and Section V provides the summary of the work.

II. Related Work

Spatial computations include convolution operations [8], [9], down-sampling, morphological, enhancement and denoising filtering [10], [11], gradient-based edge detection, and template-based correlation [12]. These computations are frequently embedded in more complex algorithms developed for image segmentation, image restoration, or object recognition and image scene understanding. They can be found in many closed- or open-source image processing packages such as Adobe Photoshop® or ImageJ [7]. However, one does not know what image processing operations are using kernel-based computations in an unknown library. According to our knowledge, the problem of detecting spatial image operations in black-box software has not been addressed yet.

The problem of efficient execution of spatial image operations has been tackled by analyzing mathematical models and by benchmarking implementations on various hardware platforms in a few published papers. For example, the authors in [9] focus on *mathematical models of convolution and their efficient implementations*. This study includes superscalar and parallel processing units (CPU, DSP, and GPU), programmable architectures (e.g. FPGA), and distributed systems (such as computer grids). It is well stated in [8] that "Basically, the convolution is a memory-bound problem, i.e. the ratio between the arithmetic operations and memory accesses is low." which is utilized in our work as well. However, the past study does not include computer cluster/cloud platforms or the MapReduce paradigm in Hadoop. Our work is also considering a broader category of spatial image computations than convolutions.

The problem of image data distribution using Hadoop has been investigated in [13] for bilateral image smoothing. The work focuses only on bilateral image smoothing as an example of *local and non-iterative*

computation because computations from the other three categories of algorithms are more difficult to implement efficiently using Hadoop. Our work differs by building an extension to Apache Hadoop that will automatically perform logical partition of images according to the estimated parameters in the detection step. Our approach goes beyond bilateral image smoothing and should be directly usable for other *local and non-iterative computations as an extension to Hadoop*.

Finally, our approach of logical image partitioning is directly related to the parallel design concepts (i.e., ghost cell pattern [14] and structured grid computational pattern [15]) that are applicable to computational grid applications by using MapReduce and geometric decomposition algorithm strategy [1]. The past work in [14] reports the same concept as ours using Message Passing Interface (MPI) while our work provides Hadoop implementation with quantitative benchmarks. Other previous studies on data locality investigated the minimum number and optimal placement of replicas [16] or opted to transfer the responsibility for optimal data locality to a job scheduler working with a uniform data replication policy in a distributed file system [17], [18].

III. Spatial Computations over Terabyte-Sized Images on Hadoop Platforms

This section describes two problems: A. The problem of detecting spatial computation and parameter estimation from a library of image processing functions. B. The problem of an optimal image partitioning schema for spatial image computations on Hadoop cluster/cloud computing platforms with respect to minimum network data transfer.

A. *Detection and Estimation of Spatial Computations in Image Libraries*

Let us assume that there exists a function F in a black-box image processing library that creates an output image from an input image by deriving each output value at location $\vec{x} \in \mathbb{Z}^D$ $(D = 2,3,\ldots)$ from a set of input image values that include the location $\vec{x}$. Fig. **2** shows an example of such an image library function that is denoted as a spatial image computation. For simplicity, we will proceed with $D = 2$.

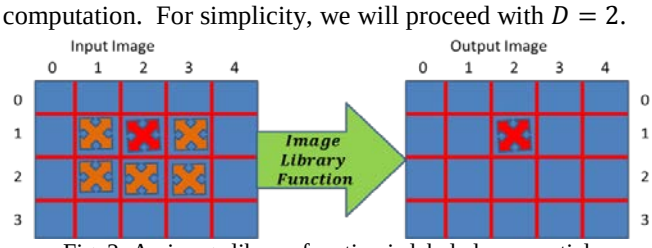

Fig. 2: An image library function is labeled as a spatial computation without or with overlap if it will compute one red pixel in output image from one or many pixels around the same location in input image.

The problems of detecting spatial computation among image library functions and estimating its spatial extent are approached by (a) forming an objective function f in equation (1) and (b) evaluating f to find subsets of input pixels $A(\vec{x})$ for which $f = 0$.

$$\boldsymbol{f}(\mathbf{A}(\vec{\mathbf{x}}), \mathbf{I_{IN}}, \mathbf{F}) = \sum_{\vec{\mathbf{x}}_\mathbf{i} \in \mathbf{A}(\vec{\mathbf{x}})} \left| \mathbf{CROP}[\mathbf{F}(\mathbf{I_{IN}}(\vec{\mathbf{x}}_\mathbf{i}))] - \mathbf{F}[\mathbf{CROP}(\mathbf{I_{IN}}(\vec{\mathbf{x}}_\mathbf{i}))] \right| \quad (1)$$

where I_{IN} is an input image modeled as a mapping $I_{IN}: \vec{x} \in \mathbb{Z}^D \rightarrow y \in \mathbb{Z}$; $F: I_{IN}(A(\vec{x})) \rightarrow I_{OUT}(\vec{x})$ is a function in an image processing library, and CROP is an image sub-setting operation $CROP: I_{IN}(\vec{x}_i \in \mathbb{Z}^D) \rightarrow I_{CROP}(\vec{x}_i \in A(\vec{x})) \subset I_{IN}$. For a given input image I_{IN} and a function F with pre-set parameters, the objective function f depends only on the parameters of $A(\vec{x})$. The parameters of $A(\vec{x})$ (i.e., input image region in TABLE 1) might vary or be constant with image location $\vec{x}$ as would be the case of a fixed size spatial kernel ($A(\vec{x}) = A$).

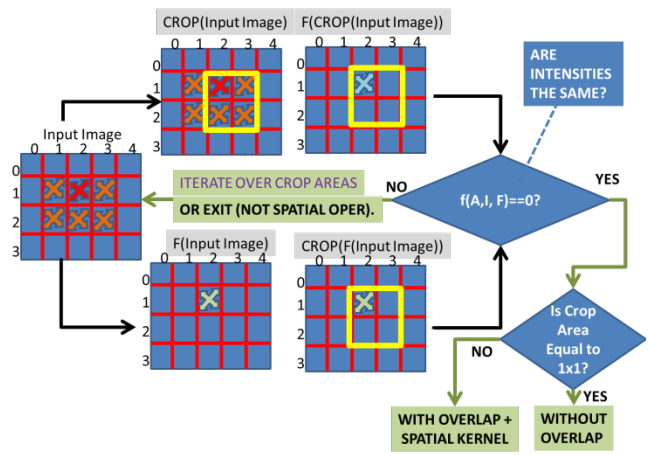

Fig. 3. Overview of the algorithm for detecting spatial overlap and estimating spatial kernel.

Fig. **3** and the algorithmic pseudo-code below illustrate the computation of an objective function and the iterative nature of the search for parameters of A.

Algorithm for detecting spatial overlap and estimating spatial rectangular kernel:

Create a set of image areas of $A = 1x1$ size (area of one pixel).

Compute $f(A, I_{IN}, F)$ for all image areas in a set.

if All values of $f(A, I_{IN}, F)$ are zero **then**

the computation is spatially local without spatial overlap and hence use <u>logical partitioning without overlap</u>

else Create a set of point-centered rectangular areas $S_{PC} = \{A_i\}$ around the *image center pixel* with varying dimensions A_x and A_y.

Compute $f(A_i \in S_{PC}, I_{IN}, F)$ for all image areas

if There exists $f(A^* \in S_{PC}, I_{IN}, F) = 0$ **then**

the computation is spatially global with spatial overlap and its rectangular kernel is A^*, thus, use <u>logical partitioning with overlap</u>.

else The computation is not spatial and hence <u>physical partitioning</u> can be directly applied.

B. *Image Partitioning Schema for Spatial Computations on Hadoop Platforms*

The information about spatial image computations can be utilized to address the problem of an optimal image partitioning schema on Hadoop cluster/cloud computing platforms with respect to minimum network data transfer. Image partitioning for parallel execution can be performed based on logical image regions rather than physical chunks of an image file. We illustrate the advantage of logical image partitioning in Fig. **4** where there is no need to exchange data between nodes during runtime. The goal is to co-localize an input image sub-region with the computation of the corresponding output value on each cluster node.

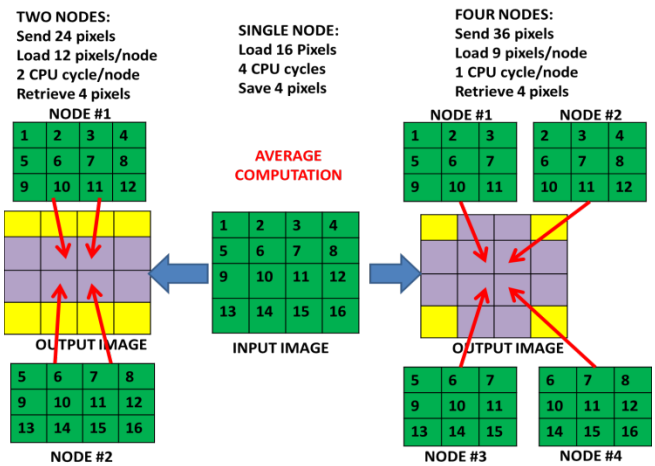

Fig. 4: Examples of image partitioning for a spatial computation of an average of 3x3 image pixels with overlapping four image regions. The computation is illustrated for a single node, and two and four distributed computational nodes with no exchange of pixels between nodes during runtime.

1) *Mathematical Framework*

In order to evaluate the runtime benefits achieved by using logical partitions for spatial image computations as illustrated in Fig. **4**, we focus on the speed-up of going from single processor desktop to a computer cluster with P processors. In general, the speed-up is defined as a ratio of the time using one processor and P processors according to equation (2). It can be viewed as a function of the number of processors or the problem/data size (strong versus weak scalability following Amdahl's and Gustafson's laws [19]).

$$S(P) = \frac{T(1)}{T(P)} \tag{2}$$

For computer clusters and cloud computing resources, the speed-up has dependencies on the number of nodes N, the node parameters $Node(P_i, M_i)$ such as the number of processors P_i and the RAM size M_i, the data parameters such as the total size of data D and data partitioning DP_k [type={physical, logical}, size= $|DP_k|$] packaged into blocks of size B for distribution across N cluster nodes, and the data transfer parameters $Network(Pa)$ of the network connecting the cluster nodes and the storage array with data. The parameter extension in equation (2) to more complex ones for a computer cluster is presented in equation (3).

$$S(P) \rightarrow S\big(N, Node(P_i, M_i), Data(D, DP_k, B), Network(Pa)\big); \quad P = \sum_{i=1}^{N} P_i \tag{3}$$

For a specific case of a computer cluster running Apache Hadoop middleware to manage computations using MapReduce, one can expand equation (3) and the speed-up dependencies can then be expressed as shown in equation (4).

$$S = \frac{T(N=1, Node(P_i=1,\ M_i),\ Data(D\ ), Network(Pa))}{T(N, Node(P_i, M_i), Data(D, DP_k, B, R), Network(Pa))/Hadoop\ )} = [T_{NET}(D) + T_{IO}(D) + n_k x T_{CPU}(DP_k)]\ / \begin{bmatrix} T_{HDFS}(D,R) + \\ T_{Map}(DP_k) + T_{Red}(DP_k) \end{bmatrix} \tag{4}$$

where in addition to the variables introduced in equation (3), R is the number of distributed image replicas, and n_k is the number of jobs per image to complete. The numerator in equation (4) includes the time needed for an execution on a single node. It consists of the network transfer time T_{NET} to read and write the data between a user disk and his/her computational node, T_{IO} to load one input image and save the results, and the time T_{CPU} to perform one of n_k computations operating on an input image sub-area DP_k. The denominator adds the times needed (a) to upload the input data to Hadoop Data File System (HDFS), replicate the data blocks across N cluster nodes and retrieve the results: T_{HDFS}, (b) to perform Map tasks: T_{Map}, and (c) to shuffle values and reduce outputs to form a final result during Reduce phase: T_{Red}. For illustration purposes, the parameters in equations (3) and (4) for the examples shown in Fig. **4** would be: N=1, 2 or 4, $Node(P_i = 1, M_i > D)$, $D = 4x4$ pixels, DP_k = [type={logical}, size = 3x3 or 3x4 pixels], B >12 pixels, $n_k = 4$, R =1, and $Network(Pa)$ are not specified.

Our logical partitioning schema focuses primarily on reducing the time for Map tasks that can be expressed in equation (5).

$$T_{Map}(DP_k) = \frac{n_k}{P} x \left(T_{IO}(DP_k) + T_{N2N}\left(\frac{1}{R}, DP_k\right) + T_{CPU}(DP_k)\right) \tag{5}$$

where T_{IO} is the time to read and write data to and from RAM, T_{N2N} is the time to transfer the pixels that are not available at a compute node to complete a computation, and T_{CPU} is the time to perform computation. Our objective is to maximize the speed-up S over physical and logical data partitions DP_k with and without overlapping pixels (denoted as ghost cells in [14]) by minimizing the communication time pertinent to input data transfers to HDFS (T_{HDFS}), and between the nodes to retrieve necessary data during the Map

phase (T_{N2N}). Both objectives are presented in the equations below.

$$\max_{DP_k} S((N, Node(P_i, M_i), Data(D, DP_k, B, R), Network(Pa))/ Hadoop) \quad (6)$$

$$\min_{DP_k} \left(\begin{array}{c} T_{HDFS}(D, R) + \\ \frac{n_k}{P} x T_{N2N}\left(\frac{1}{R}, DP_k\right) \end{array} \right) \quad (7)$$

Note: The minimization problem can also be interpreted as an evaluation of input image partitioning schemas. The execution time on a single node assumes that the input image of width/height dimensions $W_{In} x H_{In}$ and the output image of size $W_{Out} x H_{Out}$ can be handled by its RAM (loading entire input image or image regions followed by processing).

2) *Image Partitioning Schema*

Given the objective functions in equations (6) or (7), we devised an image partitioning schema for logical partitions without or with overlapping pixels into a block size $|DP_k|$ to be distributed across computer cluster nodes. For logical partitioning with additional inputs (see Fig. **1**) including a mask or a set of bounding boxes, we package pixels defined by a mask label or enclosed by a bounding box into one logical set DP_k. This turns into a bin packing problem [20] where logical sets of pixels of different cardinality $|DP_k|$ must be packed into a finite number of physical blocks N_B in a way that minimizes the number of blocks used. In general, an optimal number of blocks would be equal to the number of processors P.

Logical image partitioning for *spatial kernel-based computations* is designed with or without considerations of a kernel (sub-area) overlap for adjacent pixels (see Fig. **5**). In order to obtain image regions DP_k containing a union of kernels for neighboring pixels: $DP_k = \cup \{A(\vec{x}_j), A(\vec{x}_{j+1}), \ldots\}$, we cut each image to the desired number of regions N_B by D_x and D_y cuts along each input image dimension $N_B = (D_x + 1)x(D_y + 1)$. The blocks are either distributed directly to cluster nodes for logical partitions without overlap or are extended by the overlapping pixels for logical partitions with overlap as illustrated in Fig. **5**. The extension is performed by adding to each block K_x number of input image columns and K_y number of rows where these two numbers correspond to the additional input pixels in each dimension that are needed to compute output values using the spatial kernel $A(\vec{x}_j)$.

IV. Experimental Results

Experimental results follow the organization of Section III describing the two problems (detection/estimation of spatial computations, optimal image partitioning schema) with additional information about benchmark configurations for evaluating the image partitioning schema.

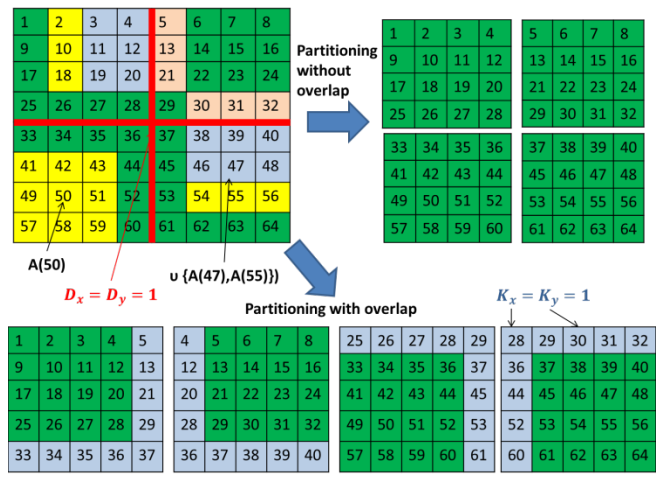

Fig. 5: Image partitioning with and without overlap by horizontal and vertical cuts. This example shows an image partition to four blocks for a kernel size 3x3 (A(50) in yellow). The values of block $\boldsymbol{K_x}$ and $\boldsymbol{K_y}$ are derived from unions of horizontal or vertical kernels of adjacent pixels.

A. *Benchmark Configurations*

1) *Terabyte-sized Image Dataset*

We experimented with 161 images of the dimensions (width ∈[22 881, 22 980]) x (height ∈[20 937, 21 123]), and 16 bits per pixel (about 1 GB per image). These images have been stitched from 127 512 files that represent 18 x 22 = 396 fields of view (FOVs), 161 time points and 2 imaging channels. One FOV is about 2.8 MB. Each stitched image covers approximately 180 mm^2 of a stem cell colony dish, over five days under both phase contrast and green fluorescence channels, with images acquired every 15 minutes. The images are stored in a TIFF file format (143 GB). Each frame has approximately 475 million pixels with 2 bytes per pixel. For benchmarking and stress testing, we used 161 images from one channel.

2) *Hardware Platform*

We have executed all Hadoop cluster benchmarks on the NIST Raritan cluster with specifications provided in Table 2. The cluster nodes had four processors (mostly Intel Xeon and Dual Core AMD) and 16 GB of RAM. We used the default Apache Hadoop configurations with 6 GB per Java process, the number of replicas R equal to two, and the block size B set to the 64 MB default size.

Table 2: Specifications of NIST Raritan computer cluster

	Specs	Cluster
Hardware	Cluster Nodes	800 computer nodes having from 2 to 16 logical cores with 4 to 32GB of RAM
	Networking	1Gbit/second
Software	Java Virtual Machine	Java version "1.7.0_17" Java(TM) SE Runtime Environment (build 1.7.0_17-b02) Java HotSpot(TM) 64-Bit Server VM (build 23.7-b01, mixed mode)

	Hadoop	hadoop-1.0.3.16
	Operating System	CentOS 5.9 Linux 2.6.18-274.3.1.el5 x86_64
	File System	Lustre parallel distributed file system for /home and ext3 for the root used by HDFS

B. *Results From Detecting and Estimating Computations with Spatial Overlap*

We tested a set of image operations from Java Advanced Imaging (JAI) [6] and ImageJ [7] using synthetic images shown in Fig. **6**. These synthetic images represent a wide variety of statistical spatial intensity arrangements for detecting local versus global properties.

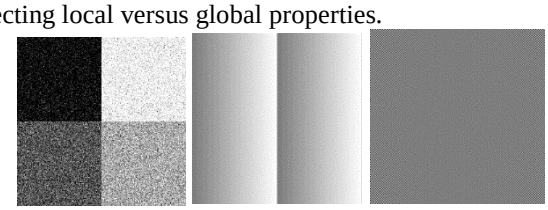

Fig. 6: Example synthetic images that represent randomness in intensity values (left), spatial gradient changes of intensity (middle), and checkerboard pattern of intensities (right).

The set of image operations was selected from the following categories.

-Unary and binary pixel operations that require only a single input pixel to compute one output pixel. Examples: addition, multiplication, absolute, and threshold.

-Neighbor operations that require several pixels around a pixel in the input image to compute one output pixel. Examples: convolution, median filter, and morphological filters.

-Other operations that do not fall into a class of spatial computations. Examples: rotation or image flipping along any axis.

The test results are summarized in TABLE 3. All results matched the ground truth.

TABLE 3: SUMMARY OF THE DETECTION AND ESTIMATION EXPERIMENTS

Image operation	Software package	Category of image operation	Recommended Image Partitioning
Multiply by constant	JAI, ImageJ	unary	Logical without overlap
Gradient magnitude	JAI	neighbor	Logical with 3 x 3 overlap
Max Filter with variable mask	JAI	neighbor	Logical with detected overlap
Image convolution	JAI, ImageJ	neighbor	Logical with detected overlap
AutoThreshold	ImageJ	unary	Logical without overlap
Morphological erosion	ImageJ	neighbor	Logical with 3 x 3 overlap
Median filter	ImageJ	neighbor	Logical with 3 x 3 overlap
Flip horizontal	ImageJ	other	Physical

C. *Runtime Results Using Image Partitioning Schemas*

Equation (7) can be evaluated experimentally over the types of image partitioning schemas and a range of image regions and kernels determining the size of DP_k[type ={physical, logical with overlap, logical without overlap}, size= $|DP_k|$]. Table **4** summarizes the parameters we have varied during experimental benchmarks. We documented the implementations of image partitioning schemas in C.1) and reported the experimental comparisons in C.2) using the configurations specified in Section A.

TABLE 4: SUMMARY OF PARAMETERS VARIED DURING EXPERIMENTAL BENCHMARKS

Parameters	Values
Number of cluster nodes N	20,40,60,80,100,120
Partitioning schema	Physical (PB), Logical without (LB) and with overlap (LBO)
Image region size	1 MB, 10 MB, 30 MB, 61 MB
Number of Map tasks	1, 2, 6
Kernel size $A = A_x * A_y$	3x3, 25x25, 51x51, 75x75, 101x101
Data size	40, 161 images (1 GB per image)

1) *Parallel Distributed Implementations*

We implemented the logical partitioning with and without overlap as an extension to the existing physical partitioning schema in Apache Hadoop following Section III.B. During the push of an image into HDFS from a file storage system, the image is subdivided into regions with or without overlapping pixels. An image region is converted into an image record which is defined as a key/value pair. The key is generated from the file name of the whole image. The value holds intensities of an image region with or without overlapping pixels, and the region position in the coordinate system of the input image. Multiple image records are stored using Hadoop Sequential File Format (i.e., SequenceFile objects) to avoid problem of many small files in Hadoop HDFS.

During the Hadoop Map phase, a spatial image computation is applied to every image record to generate an intermediate image record represented by a new key/value pair. During the Hadoop Reduce phase, values in intermediate records with the same key are shuffled and sorted among the worker nodes. The Reduce function retrieves computed output image pixels/regions from the same input image and produces the final output image using the information associated with image records.

Table **5** provides details of Hadoop MapReduce implementations for the three image partitioning schemas including physical, logical without overlap, and logical with overlap.

TABLE 5: SUMMARY OF HADOOP IMPLEMENTATIONS FOR THREE IMAGE PARTITIONING SCHEMAS INCLUDING PHYSICAL, LOGICAL WITHOUT OVERLAP, AND LOGICAL WITH OVERLAP

Hadoop Implementation		Map	Reduce
logical image partitioning **WITH overlap**	**Input**	Image records (key/value pairs) and location of all image regions	Key and list of values for each input image
	Output	Image records (key/value pairs)	One output image per input image
	Function	image computation on image records	Stitch all computed image records with the same key. Write output as a single image to HDFS
logical image partitioning **WITHOUT overlap**	**Input**	Image records (key/value pairs) and location of all image regions	Key and list of values for each input image
	Output	Image records (key/value pairs)	One output image per input image
	Function	for each image record: -Retrieve image region position -Load all neighboring regions of the image record from HDFS -Form image regions with overlapping pixels along its borders -Execute image computation on a region and extract computed region without overlap as an output image record	Stitch all computed image records with the same key. Write output as a single image to HDFS
physical partitioning of **NON SPLIT** images	**Input**	Key: image number Value: entire image loaded from HDFS	**None**
	Output	Key: write computed entire image directly to HDFS Value: computed region	**None**
	Function	image computation on entire image	**None**

2) Experimental Comparisons of Logical and Physical Image Partitioning Schemas

All experimental benchmarks are obtained using morphological dilation operation applied to the stem cell images described in Section A.1). Each compute node came with 4 processors and 16 GB RAM according to the cluster specifications in Section A.2). The physical partitioning schema used 64 MB block size for pushing the data to Hadoop HDFS. The block size for the two logical partitioning schemas varied according to Table **4**.

a) Strong and weak scaling

We have varied the input image size and the number of nodes to benchmark the scaling performance according to weak and strong scaling assumptions [19]. Fig. **7** and Fig. **8** show runtime dependency for morphological dilation with a kernel size A=101x101 on the number of cluster nodes for 40 and 161 images and multiple partitioning configurations. The number of file replication R is two.

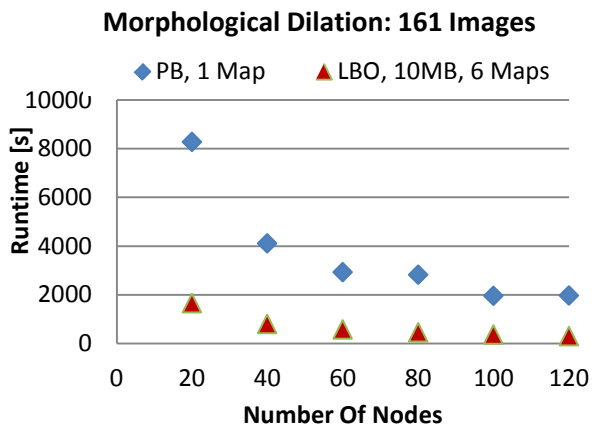

Fig. 7: Runtime as a function of the number cluster nodes. The logical partitioning with overlap (LBO) into 10MB image regions outperforms the physical partitioning (PB) for 161 images.

Fig. **7** illustrates close to linear scaling for Hadoop logical partitioning with overlap (LBO) for up to 120 cluster nodes with the total of 480 processors. The average relative speed-up of 5.36 is computed as the ratio of time averages over the collected data points using physical (PB) and logical partitioning with overlap (LBO) and 10 MB image regions. The difference in the runtimes is due to not only the logical partitioning but also the ability to run up to 6 Map tasks concurrently. In addition, the RAM requirement to computations using PB is 8 GB in comparison to less than 1 GB using LBO.

Fig. **8** shows that the Hadoop physical partitioning (PB) configuration does not scale for the number of cluster nodes larger than 60. There are idle nodes and the cluster is unbalanced. The lack of scaling could also be due to the executions of duplicated tasks on big images for failed tasks and waiting for the execution of the last task. The scalability of the computation clearly benefits from splitting the big images into image regions in a distributed computer cluster environment.

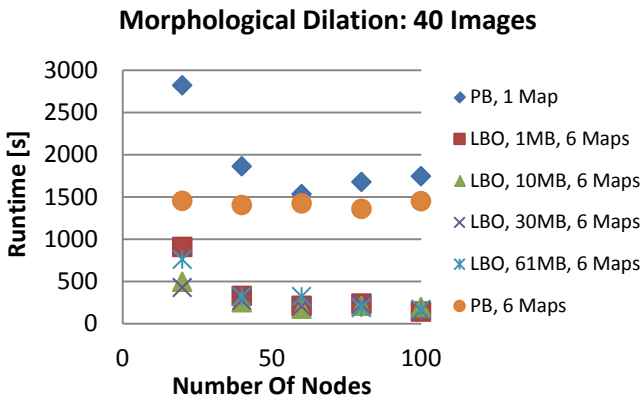

Fig. 8: Runtime as a function of the number cluster nodes. The logical partitioning with overlap (LBO) outperforms the physical partitioning (PB) for 40 images regardless of the image region size.

b) Logical partitioning schemas

We evaluated the runtime difference between logical partition with and without overlap as a function of the number of cluster nodes in Fig. **9**. The benchmarks are collected for morphological dilation with a kernel size A=101x101 on varying number of cluster nodes. The logical partitioning size of an image region was fixed to 10 MB and six Map tasks were executed on each node. One can compute a runtime speed-up as a ratio of logical partitioning with overlap over without overlap from the data shown in Fig. **9**. The average speed-up over varying number of cluster nodes is 3.14 (minimum=2.2 and maximum=5.4). We have also analyzed percentages of data local tasks for the 40 processed images based on log files. The difference in percentages ranges between 15 and 65 (LBO, 1 MB versus PB) and corresponds to the extra time spent on node to node data transfer T_{N2N}.

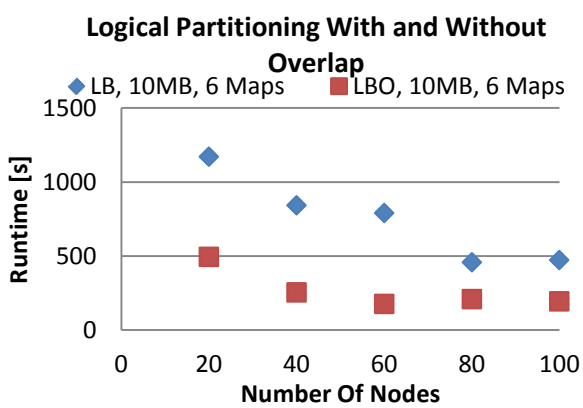

Fig. 9: Comparison of runtimes using logical partitioning with and without overlap.

In addition, we explored the dependency of runtime on the kernel size of an image morphological dilation. Fig. **10** shows the benchmarks for a configuration processing 40 image files on 40 cluster nodes and a fixed image region size of 10 MB and 2 Map tasks per node. We concluded that LBO always has lower runtime. The runtime difference is almost constant with the kernel size due to the transfer time needed by LB to bring a whole data block containing the missing pixels to the compute node.

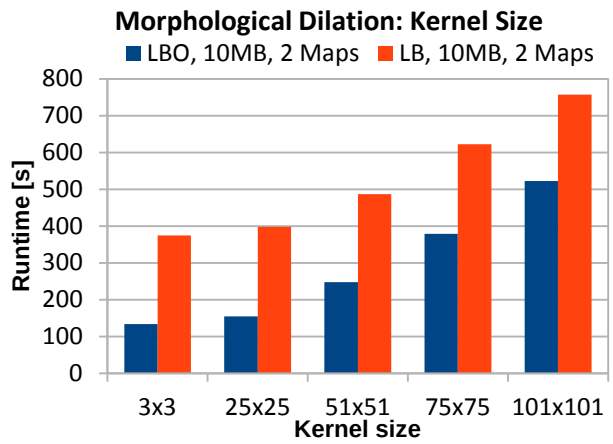

Fig. 10: Runtime dependency on the size of a morphological dilation kernel for the two logical partitioning schemas with and without overlap. The benchmarks are collected using 40 cluster nodes, 10 MB image regions and 2 Map tasks.

V. Summary

We have analyzed a class of spatial image computations applied to terabyte-sized images and executed on a Hadoop computer cluster platform. We addressed two problems that would lower the barrier for bench scientists to process large size images by (a) detecting spatial image computations in a library of image processing functions, and (b) partitioning image data for spatial image computations on Hadoop cluster/cloud computing platforms in order to minimize network data transfer. Our theoretical framework focused on formulating both problems as estimation problems, and evaluating various image partitioning configurations. The experimental part documented accuracy and runtime performance of multiple image partitioning schemas for morphological dilation used as an example of a spatial image processing operation. The results for the detection problem demonstrated 100% accuracy in detecting spatial computations. The results with various image partitioning schemas yielded a significant speed-up (5.36 and 3.14) of the computations on Hadoop clusters when comparing physical or logical partitioning without overlap and logical partitioning with overlap.

In the near future, we plan to benchmark other image processing computations and disseminate the Hadoop extension to the image processing community. Overall, there are still unanswered questions about the distribution of image data access (uniform or skewed), temporal locality in data access, and how much of multi-dimensional image data is collocated when being accessed. In order to understand the relationship between data parallelism and computational

efficiency, one has to examine specific degrees of dependency among data points and data access patterns for each class of image processing computations.

ACKNOWLEDGMENT

This work was sponsored by NIST as a part of the Computational Science in Biological Metrology project. We would like to acknowledge all project team members for their contributions.

DISCLAIMER

Commercial products are identified in this document in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the products identified are necessarily the best available for the purpose.

REFERENCES

[1] K. Keutzer, L. Berna, G. Timothy, and A. Beverly, "A Design Pattern Language for Engineering (Parallel) Software : Merging the PLPP and OPL Projects," in *Proceeding ParaPLoP '10 Proceedings of the 2010 Workshop on Parallel Programming Patterns*, 2010, pp. 1–8.

[2] T. White, *Hadoop: The Definitive Guide MapReduce for the Cloud*, 3rd ed. O'Reilly Media, 2012, p. 528.

[3] O. O. Malley, "Hadoop Benchmarking," in *Workshop on Big Data Benchmarking*, 2012, no. May, pp. 1–13.

[4] Yahoo!, "Hadoop Tutorial from Yahoo!," *On-Line Tutorial*, 2013. [Online]. Available: http://developer.yahoo.com/hadoop/tutorial/module4.html. [Accessed: 14-May-2013].

[5] Xyratex, "Using Lustre with Apache Hadoop," 2010. [Online]. Available: http://wiki.lustre.org/images/1/1b/Hadoop_wp_v0.4.2.pdf. [Accessed: 31-May-2013].

[6] JavaSoft, "Programming in Java Advanced Imaging," Sun Microsystems, 901 San Antonio Road Palo Alto, CA 94303 USA, 1.01, 1999.

[7] W. Rasband, "ImageJ & Fiji & ImageJA & ImageJ2, Computer Program," 2013. [Online]. Available: http://rsbweb.nih.gov/ij/. [Accessed: 15-May-2013].

[8] D. Svoboda, "Efficient Computation of Convolution of Huge Images," in *Image Analysis and Processing ICIAP 2011*, G. Maino and G. Foresti, Eds. Lecture Notes in Computer Science; Springer-Verlag, 2011, pp. 453–462, Vol. 6978.

[9] P. Karas, D. Svoboda, K. Pavel, and S. David, "Algorithms for Efficient Computation of Convolution Algorithms for Efficient Computation of Convolution," in *In Design and Architectures for Digital Signal Processing*, 1st ed., Rijeka, Croatia: IN-TECH, Open Science - Open Mind, 2013, pp. 179–207.

[10] I. N. Bankman, *Handbook of Medical Image Processing and Analysis*, 2nd ed. Burlington, MA: Academic Press; Elsevier, 2009, p. 984.

[11] J. C. Russ, *The Image Processing Handbook*, Sixth. Boca Raton, FL: CRC Press, Taylor& Francis Group LLC, 2011, p. 839.

[12] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York: John Wiley & Sons, 2011, p. 637.

[13] K. Potisepp, "Large-scale Image Processing Using MapReduce," Tartu University, 2013.

[14] F. B. Kjolstad and M. Snir, "Ghost Cell Pattern," in *Proceedings of the 2010 Workshop on Parallel Programming Patterns - ParaPLoP '10*, 2010, pp. 1–9.

[15] K. Gonina, H. Bayandorian, and E. Strohmaier, "Structured Grid Computational Pattern," *A Pattern Language for Parallel Programming ver2.0*, 2010. [Online]. Available: http://parlab.eecs.berkeley.edu/wiki/patterns/patterns. [Accessed: 16-Jun-2014].

[16] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, "CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster," in *IEEE Int'l Conf. Cluster Computing (CLUSTER)*, 2010, pp. 188–196.

[17] P. Nguyen, T. A. Simon, M. Halem, D. Chapman, and Quang Le, "A Hybrid Scheduling Algorithm for Data Intensive Workloads in a Map Reduce Environment," in *5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC2012)*, 2012, pp. 161–167.

[18] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay Scheduling : A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," *Eur. Conf. Comput. Syst.*, pp. 1–14, 2010.

[19] J. L. Gustafson, "Reevaluating Amdahl's Law," *Commun. ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[20] B. Xia and Z. Tan, "Tighter bounds of the First Fit algorithm for the bin-packing problem," *Discret. Appl. Math.*, vol. 158, no. 15, pp. 1668–1675, Aug. 2010.