Published in final edited form as:

Concurr Comput. 2018 July 25; 30(14): . doi:10.1002/cpe.4425.

Cooperative and out-of-core execution of the irregular wavefront propagation pattern on hybrid machines with Intel® Xeon Phi™

Jeremias Gomes[1], **Alba C. M. A. de Melo**[1], **Jun Kong**[2], **Tahsin Kurc**[3], **Joel H. Saltz**[3], and **George Teodoro**[1,3]

[1]Department of Computer Science, University of Brasília, Brasília-DF, Brazil

[2]Department of Biomedical Informatics, Emory University, Atlanta, GA, USA

[3]Department of Biomedical Informatics, Stony Brook University, Stony Brook, NY, USA

Summary

The Irregular Wavefront Propagation Pattern (IWPP) is a core computing structure in several image analysis operations. Efficient implementation of IWPP on the Intel Xeon Phi is difficult because of the irregular data access and computation characteristics. The traditional IWPP algorithm relies on atomic instructions, which are not available in the SIMD set of the Intel Phi. To overcome this limitation, we have proposed a new IWPP algorithm that can take advantage of non-atomic SIMD instructions supported on the Intel Xeon Phi. We have also developed and evaluated methods to use CPU and Intel Phi cooperatively for parallel execution of the IWPP algorithms. Our new cooperative IWPP version is also able to handle large out-of-core images that would not fit into the memory of the accelerator. The new IWPP algorithm is used to implement the Morphological Reconstruction and Fill Holes operations, which are operations commonly found in image analysis applications. The vectorization implemented with the new IWPP has attained improvements of up to about 5× on top of the original IWPP and significant gains as compared to state-of-the-art the CPU and GPU versions. The new version running on an Intel Phi is 6.21× and 3.14× faster than running on a 16-core CPU and on a GPU, respectively. Finally, the cooperative execution using two Intel Phi devices and a multi-core CPU has reached performance gains of 2.14× as compared to the execution using a single Intel Xeon Phi.

Keywords

Fill Holes; Intel® Xeon Phi™; Irregular Algorithm Propagation Pattern; Morphological Reconstruction

1 | INTRODUCTION

This paper presents efficient algorithms for the Irregular Wavefront Propagation Pattern (IWPP)[1] on Intel's Xeon Phi (Intel Phi) co-processors. Our work is motivated by the

Correspondence: George Teodoro, Department of Computer Science, University of Brasília, Brasília-DF, 70910-900, Brazil. teodoro@unb.br.

ORCID

George Teodoro http://orcid.org/0000-0001-6289-3914

computational requirements of segmenting nuclei in whole slide tissue images (WSIs). WSIs enable the quantitative study and characterization of the relationship between tissue morphology at the sub-cellular level[2] and disease mechanisms. Cancer, for example, manifests itself through changes in nuclear morphology. Characterizing these changes and correlating the characterization with clinical and genomic data can result in a better understanding of disease onset and progression, leading to improvements in methods for assessing response to treatment. While a WSI contains rich information, a typical analysis pipeline with normalization, object segmentation, and feature computation stages can take hours on a single CPU core; with modern digitizing microscope scanners, the pixel resolution of a WSI can range from 50K×50K to over 100K×100K pixels. The high computation demand poses as a major challenge to studies involving large datasets of WSIs and has motivated our group to develop parallel computing techniques and systems to accelerate this class of applications using high performance machines.[1] Some core operations in image analysis applications share a core computation structure called Irregular Wavefront Propagation Pattern (IWPP).[1] For instance, segmentation operations that use the IWPP include: Morphological Reconstruction,[3] Fill Holes,[4] H-minima/maxima,[4] Watershed,[5] and Distance Transform.[6]

The Intel Phi is becoming an increasingly popular and powerful processor and, as a consequence, is an attractive platform for medical image analysis applications. Efficient implementation of the IWPP on the Intel Phi is a complex task. The IWPP performs computations by carrying out wave propagations from source grid points (image pixels) to neighbor points. The waves are irregular in shape and propagations may occur into several points of the domain. Moreover, the wave propagations are data dependent, and interaction between waves may change the waves' directions. The irregularity of the waves propagations makes the IWPP a challenging pattern for efficient parallel execution, whereas the use of atomic instructions in the original IWPP creates additional difficulties for deploying it on the Intel Phi.[7] The Intel Phi achieves its maximum performance when its SIMD instructions are used. However, atomic instructions are not included in the Intel Phi SIMD instruction set. In addition, hybrid machines with CPUs and one or more co-processors are becoming more popular, and using all the available computing capacity (CPU cores and co-processors) in a coordinated manner is critical to minimize application execution time.

In this paper, we address these challenges of IWPP with new algorithmic strategies to efficiently execute it on the Intel Phi and approaches to decompose the algorithm execution into multiple processors. This paper extends our preliminary work[8] with the introduction of capabilities for cooperative execution, the ability to process large out-of-core images, and a detailed evaluation of the proposed optimizations. The contributions of our work can be summarized as follows:

- We developed a new parallel execution strategy of the IWPP that uses only non-atomic SIMD instructions and, as a consequence, fully benefit from the Intel Phi.
- We extended the baseline IWPP to support processing of images using multiple Intel Xeon Phis and CPUs in cooperation. This version of IWPP also enables the

computation of large images using the co-processor, even when the co-processor memory is not sufficient to store the entire image data (out-of-core).

- We have evaluated our propositions using the Morphological Reconstruction and Fill Holes operations using a machine equipped with two Intel Phi co-processors and a multi-core CPU. The results have shown that the new vectorized version is up to 5.6× faster than the original IWPP on the Intel Phi. It also is about 6.2× faster than the multi-core (16-core) CPU version. In addition, the execution using a multicore CPUs and two Intel Xeon Phi devices cooperatively resulted in a speedup of 2.2× vs a single Intel Xeon Phi. These performance gains demonstrate the feasibility of using the Intel Xeon Phi to rapidly execute image analysis operations.

We should note that the optimizations proposed in this work for efficient use of SIMD instructions and thread-level parallelism will benefit other architectures. Modern processors, such as the Intel Skylake, are also equipped with wide 512-bit vector processing units (SIMD units) and have a large number of computing cores. The rest of this paper is organized as follows. The motivation application is presented in Section 2. The original IWPP is discussed in Section 3. The new and vectorized version of the IWPP is described in Section 4. The strategy to cooperatively execute IWPP algorithms using multiple Intel Phi processors and multi-core CPUs is discussed in Section 5. The experimental evaluation is shown in Section 6, and the related work is detailed in Section 7. Finally, the conclusions and future work are discussed in Section 8.

2 | BACKGROUND

This section provides an overview of the image analysis application that motivates our work and presents details of the Intel Phi architecture. Our work targets the normalization, segmentation, and feature computation stages for efficient execution because these are the most compute intensive stages of our applications. Additional analysis phases that follow these stages often deal with aggregate information.[9] In the rest of this section, we describe the motivating application domain (Section 2.1), the details of the fine-grain operations used in each application stage in order to present the impact of accelerating IWPP to the entire application workflow (Section 2.2), and discuss the Intel Phi processor architecture details and the used programming models (Section 2.3).

2.1 | Microscopy image analysis

The use of high-resolution microscopy images facilitates the study of several diseases at cellular and sub-cellular levels. The investigation of morphological changes in tissue images at this scale can reveal important information about the disease mechanisms and, as a consequence, may help on understanding response to treatment and complement genomic or clinical data.[2,9] Current digital microscopy scanner are able to collect images of tissue specimens with a very high resolution. An uncompressed 8-bit color image with three-channels may have over 50 GB, whereas a scanner can capture in the order of hundreds images per day. This capacity of modern scanner has led to an increasing availability of images in both public repositories and health care facilities. For instance, The Cancer

Genome Atlas (TCGA) contains over 30,000 WSIs. As such, we expect that moderate size studies using WSIs will handle thousands of images in a near future, creating a demand for more efficient image analysis workflows and operations.

The compute intensive stages of the image analysis workflow are presented in Figure 1 along with the main internal operations used in each stage. The normalization is responsible for correcting artifacts of the image scanning process, which, for instance, may result in images with different color intensities. The segmentation is executed to extract cells and nuclei contour information, and the feature computation stage is further applied to extract a set of shape and texture features for each of the objects identified in the segmentation. The resulting feature vector may have in the order of about 50-70 features per object. One of the main challenges with the execution of the analysis workflow is the high computation demand. Processing of a single image may take hours on a single CPU core. Thus, the internal operations in computation intensive analysis stages are good candidates for execution on co-processors to speed up application execution times.[10]

In the next section, we briefly describe the internal operations of the workflow, and the importance of the IWPP-based operations to the application execution time.

2.2 | Details of core operations and impact of IWPP acceleration on analysis workflow

The fine-grain operations used in each of the compute intensive stages are shown in Table 1. Additional details on the operations used by each of the application stages may be found in our previous work.[2,7] The "parallelism" column in the table shows the type of parallel implementation (ie, data parallel, IWPP, and object parallel) suitable for each operation.

The operations in the normalization stage mostly employ data parallelism as they transform images from different formats and normalize the color intensity. All these operations can benefit from data or loop parallelism and can be easily parallelized for a multi-core CPU, a GPU, or an Intel Phi. On the Intel Phi, the Intel Compiler is able to automatically identify parallelism for these operations as well as for the data-parallel operations in the feature computation stage.

The operations in the segmentation stage are still applied to image pixels, but most of them employ the IWPP. Parallel execution of the IWPP requires active data elements (pixels), which are non-uniformly and dynamically distributed in the image domain, to be processed in parallel. The complex and dynamic wavefront propagation pattern makes efficient parallelization a challenge. Current auto-parallelizing compilers (including the Intel Compiler) fail to generate efficient parallel versions of these operations. Even manual parallelization is very difficult with the SIMD instructions available in the Intel Phi.

The feature computation stage employs the IWPP, Data, and Object parallelisms. The object parallelism is used when statistics are computed for multiple objects (cell or nuclei) identified in the segmentation phase. The number of segmented objects in an input image or image tile is typically very high. This allows for partitioning of the set of segmented objects for concurrent and independent execution. This bag-of-tasks style execution strategy is efficient and simplifies the deployment of the codes in a parallel setting.

As shown in the table, the IWPP occurs in 5 out of 7 operations in the segmentation stage and 1 operation in the feature computation stage. Thus, the IWPP represents over 85% of the computation cost in the segmentation stage and about 40% of the entire application. This demonstrates the relevance of the IWPP in image analysis; an efficient implementation of the IWPP is critical to accelerate the entire image analysis application.

2.3 | Intel Xeon Phi

The Intel Phi co-processor is based on the Intel Many Integrated Core (MIC) architecture, which consists of many simplified and power efficient computing cores equipped with a 512-bit vector processing unit (SIMD unit). In this architecture, the computing cores are replicated to create multicore processors that are placed in a high performance bidirectional ring network with fully coherent L2 caches. The MIC architecture combines features of general-purpose CPUs and many-core processors or accelerators to provide an easy to program and high-performance computing environment.[12] It is based on a x86 instruction set and supports traditional parallelization tools and communication models, such as OpenMP (Open Multi-Processing), Pthreads (POSIX Threads Programming), and MPI (Message Passing Interface).

This work used Intel Phi processors from the Knights Corner (KNC) and Knights Landing (KNL) generations, as presented in Table 2. These devices can be deployed as co-processors attached to the CPU through a PCIe channel. In this setup, the Intel Phi runs an application in two modes: (i) the native mode allows for the user to directly access the co-processor to run the application entirely from the device; (ii) in the offload mode, on the other hand, the application is executed in the host CPU, but sections of the computation selected by the developer using pragma commands are offloaded for execution within the co-processor. The KNL can also be deployed as a standalone or bootable processor. This leads to significant improvements for data-intensive applications as overheads of transferring data from CPU and co-processor do not exist.

The characteristics of the Intel Phi devices used in this work are presented in Table 2. Both KNC processors have the same number of computing cores, but differ in terms of the processors clock frequency. The 7250 KNL, on the other hand, has more computing cores and a higher clock frequency, which leads to higher computing capacity as compared with the previous Intel Phi generation. Both generations support 512-bit SIMD instructions. The KNC was built using the “Intel Initial Many Core Instructions” (IMCI). The KNL, on the other hand, supports the AVX-512 instruction set, which is also used in mainstream processors. Although IMCI and AVX are not equivalent, we can easily map our implementation from one instruction set to the other because we use standard SIMD instructions in our codes.

3 | IRREGULAR WAVEFRONT PROPAGATION PATTERN (IWPP) IN IMAGE ANALYSIS

This section presents the IWPP and illustrates it in two image analysis operations: Morphological Reconstruction and Fill Holes. The IWPP consists of a set of propagation transformations applied on pixels of interest in an image. These propagations may result in a pixel disseminating its value to a neighborhood. The neighborhood $N_G(p)$ of a pixel p is defined using a structuring element G centered in p. A pixel q is said to be neighbor of p if $(q, p) \in G$. Typical G structures include 4-/8-connected square grids, and the domain of image pixels (D_I) is rectangular. Image pixel values could be 0 or 1 in the binary case, gray levels, or even be defined in a continuous scale.

The IWPP modifies the input by performing data propagations starting at a set of data elements (active elements) that satisfy a propagation condition. These elements work as sources of waves that, during computation, may modify neighbor pixels by disseminating their values. If a pixel receives a wave propagation, it becomes part of the wavefront and is inserted in the set of active elements, whereas a pixel that has just propagated information to the neighborhood becomes inactive. There may be several waves active in the image domain during execution. The propagation of a wave may interfere with other waves' propagations. This interference may create new waves or modify their directions.

An important property of the IWPP is that only active elements contribute to the output and, as a consequence, should be processed. Thus, scanning the entire input domain to process each element, as is employed in many regular image processing operations, will be inefficient in the IWPP. A fast execution of this pattern involves storing and keeping track of active elements. This is accomplished in our implementation by using queues or sets to save active elements. The IWPP was first presented in the work of Teodoro et al[1] and Algorithm 1 shows its computation structure.

Algorithm 1

Irregular Wavefront Propagation Pattern (IWPP)

```
Input: D: data elements in a multi-dimensional space
Output: D: stable set with all propagations reached
1:  {Initialization Phase}
2:  $S \leftarrow$ subset active elements from $D$
3:  {Wavefront Propagation Phase}
4:  while $S \neq \emptyset$ do
5:      Extract $e_i$ from $S$
6:      $Q \leftarrow N_G(e_i)$
7:      while $Q \neq \emptyset$ do
8:          Extract $e_j$ from $Q$
9:          if $PropagationCondition(D(e_i), D(e_j))$ = true then
10:             $D(e_j) \leftarrow max/min(D(e_i), D(e_j))$
11:             Insert $e_j$ into $S$
12:         end if
13:     end while
14: end while
```

The IWPP algorithm receives as input the image or data domain in which the computation should be carried out and returns the data after all possible propagations were computed. In its first phase, some preprocessing computations (not shown) may be carried out according to the actual algorithm implemented in IWPP, and the wavefront or set of active elements are identified and stored in S (line 2). After that, the propagation phase is executed in lines 4-14. In this stage, while the set of active elements is not empty or all possible propagations were computed, a given active element (e_i) is removed from S and the neighbor elements (Q) of e_i are selected according to the structuring element G chosen by the user. Further, for each neighbor e_j of e_i (lines 7-13), the propagation condition from e_i to each e_j is evaluated (line 9). If the condition is true, e_i propagates its information to e_j and the latter becomes part of the active set of elements. The wavefront propagations are expected to be commutative and atomic. The next sections present the instantiation of this pattern for two algorithms used to evaluate our optimizations.

3.1 | Morphological reconstruction

Morphological reconstruction is an important operation frequently used in image segmentation.[3,13] It can be employed to derive information about shapes available in an image, which could be, for instance, letters of scanned documents, cells in scanned tissue images, or objects as cars or buildings in a satellite image. The morphological reconstruction can extract, for instance, peak points or areas, bright regions, etc. The computation in this operation is performed following a flood-filling strategy, which starts from marker images and proceeds until it reaches values of another image called mask. Figure 2 illustrates the process of morphological reconstruction in binary and grayscale images. The marker intensity profile is propagated spatially but is bounded by the mask image intensity profile in both examples. In the binary case, the operations removed a non-marked (seeded) object and identified/removed peaks in the grayscale image.

Algorithm 2

Morphological Reconstruction Algorithm

```
Input: I:mask image, J: marker image
Output: J:reconstructed image
1:  {Initialization Phase}
2:  Scan $I$ and $J$ in raster order.
3:     Let $p$ be the current pixel
4:     $J(p) \leftarrow (max\{J(q), q \in N_G^+(p) \cup \{p\}\}) \wedge I(p)$
5:  Scan $I$ and $J$ in anti-raster order.
6:     Let $p$ be the current pixel
7:     $J(p) \leftarrow (max\{J(q), q \in N_G^-(p) \cup \{p\}\}) \wedge I(p)$
8:     if $\exists q \in N_G^-(p) \mid J(q) < J(p)$ and $J(q) < I(q)$
9:        queue.enqueue(p)
```

```
10: {**Wavefront Propagation Phase**}
11: while queue.empty() = false do
12:         $p \leftarrow$ queue.dequeue()
13:         for all $q \in N_G(p)$ do
14:             if $J(q) < J(p)$ and $I(q) \neq J(q)$ then
15:                 $J(q) \leftarrow min\{J(p), I(q)\}$
16:                 queue.enqueue(q)
17:             end if
18:         end for
19: end while
```

The morphological reconstruction implementation using IWPP is presented in Algorithm 2. As shown, this algorithm has an initialization phase in which raster and anti-raster scans are performed to compute initial propagations. The N_G^+ and N_G^- denote the set of neighbors in $N_G(p)$ that are reached before and after touching pixel p during a raster scan. These initial passes on the image are part of the most efficient algorithm for this operation,[3] and the reasoning for applying them is that they may be able to resolve a large number of propagations and reduce the amount of work performed in the actual IWPP phase. After these scans, the pixels to satisfy the propagation condition are inserted into the wavefront set, which is implemented into our algorithm using a regular First-In, First-Out queue. Further, during the wavefront phase (lines 11-19), the algorithm continues by removing an element p from the wavefront and trying to propagate the value of p to each of its neighbors q. This condition checks whether the value of p ($J(p)$) is higher than the value of q ($J(q)$) and if the value of q has not reached its limit or the mask value ($J(p) \neq I(q)$). If the condition is true, the information of p is propagated to q, and q is inserted into the set of active elements.

3.2 | Fill holes

The fill holes operation is also widely used in image analysis to remove/fill unwanted holes in objects during image segmentation steps, but it may also be employed to make the image more homogenized and smooth pixel value differences.[4] Similarly to the morphological reconstruction, the fill holes will perform a flood-filling, for instance, in a binary image that stops when the boundaries of objects or mask values are reached. In this process, holes are filled in binary images as dark areas inside objects receive propagation from surrounding white areas, whereas lighter areas in grayscale images tend to propagate to darker ones.

Figure 3 presents an example of the fill holes use for binary and grayscale images. As may be observed in the figure, in the binary image case, which was generated here by applying a threshold-based segmentation on the original image, the execution of this operation will lead to a resulting image without holes inside connected components. The use of the same operation in a grayscale image, shown in the bottom part of the same figure, results in a homogenized image in which lighter values propagate over dark areas. The use of this operation is diverse in image processing workflows, and the implementation is similar to that present for the morphological reconstruction, except that, in the initial phase, the fill holes invert the input marker image to use it as a mask during the wavefront propagation step.

4 | THE DESIGN AND IMPLEMENTATION OF IWPP FOR THE INTEL XEON PHI

This section presents our approach to efficiently execute the IWPP in the Intel Phi co-processor. As previously discussed, the original IWPP requires the use of atomic instructions, which are not available in the Intel Phi SIMD instruction set. The use of vector instructions in the Intel Phi is essential to attain high performance because of the processors' wide 512-bit vector registers and the associated vector instructions. Therefore, in order to fully utilize the computing device for IWPP, we have proposed a novel algorithmic approach that eliminates the use of atomic instructions and can take advantage of the SIMD instruction to accelerate the IWPP in the Intel Phi.

The rest of this section is organized as follows. In Section 4.1, we present the new IWPP in its sequential and non-vectorized form. The vectorized approach for the new IWPP is discussed in Section 4.2. Further, in Section 4.3, we propose the extension of the vectorized IWPP with the addition of thread-level parallelism and, in Section 4.4, we present an additional execution strategy that reduces the amount of recomputation in the algorithm due to the flood-filling pattern employed by the IWPP.

4.1 | New sequential IWPP

The main problem that prevented the use of vector instructions into the original IWPP, presented in Algorithm 1, was the use of atomic instructions to avoid race conditions in its parallel version. The race condition of the original IWPP occurs when multiple active elements concurrently propagate their values to the same element in the image domain, as presented in Figure 4(A). In this case, it is necessary to assert that the *PropagationCondition* evaluation and the actual data propagation are atomic.

To address this problem, in the new sequential IWPP algorithm, we have inverted the propagation direction. In this strategy, elements that receive propagation are first identified and become active to update their own values in a second step. As such, each of the identified elements will become a sink that captures the information from the neighborhood, as presented in Figure 4(B). In this case, there will not exist two active elements in the neighborhood trying to update a third common element in the domain with different values in the thread-level parallel version of the algorithm. This approach allows us to eliminate the use of atomic operations. However, one may notice that it is possible for a given element to be inserted into the set of active ones (information receivers) multiple times. This would potentially create another race condition resulting in multiple threads processing the same element. However, as discussed in detail in Section 4.3, this will not require the use of atomic instructions.

Algorithm 3

Redesigned IWPP

```
Input: D: data elements in a multi-dimensional space
```

```
Output: D: stable set with all propagations reached
1:  {Initialization Phase}
2:  $S \leftarrow$ subset active elements from $D$
3:  {Wavefront Propagation Phase}
4:  while $currWave.empty() = $ false do
5:    {Identification of elements receiving propagation}
6:    for all $p \in currWave$ do
7:      for all $q \in N_G(p)$ do
8:        if $PropagationCondition(D(q), D(p)) = $ true then
9:          $nextWave.insert(q)$
10:       end if
11:     end for
12:   end for
13:   {Propagation}
14:   for all $p \in nextWave$ do
15:     for all $q \in N_G(p)$ do
16:       $D(p) \leftarrow max/min(D(q), D(p))$
17:     end for
18: end for
19: $currWave \leftarrow nextWave$
20: $nextWave \leftarrow \emptyset$
21: end while
```

The new redesigned IWPP is shown in Algorithm 3. The wavefront propagation phase, which is the main focus for our optimizations, is divided into two steps in the new algorithm. First, the active elements (stored into the *currWave* container) are processed in order to identify those data elements that will receive propagation (lines 6 to 12). The ones identified as information receivers are inserted into a new set of active elements (*nextWave*), which also corresponds to those elements that will be part of the wave in the next propagation cycle that starts in line 4. Further, the actual propagation takes place in lines 14 to 18. During this stage, each element identified as an information receiver will verify the neighborhood element from which it should receive information. Finally, after the propagation is performed, the next set of active elements (*currWave*) receives the set containing the information receivers in the current iteration of the wavefront propagation. This process continues until the wavefront becomes empty and stability is reached.

4.2 | Vectorized version of new IWPP

This section describes our approach to vectorize the new IWPP algorithm presented in the previous section. The vectorization focuses in the wavefront propagation phase and is presented in Algorithm 4. This process starts with changes in the identification of elements receiving propagations (lines 9 to 16). In this step, the addresses of the active elements in the wavefront (*currWave*) are first loaded into a vector register (vec_p - line 10) and replicated into multiple positions of this vector. The number of positions is the same as the size of the neighborhood considered. If the vector register is sufficient to store the neighborhood of multiple active elements, more than one can be processed into a single pass. For sake of simplicity, we present the algorithm as if a single p was processed per loop iteration.

Algorithm 4

Nfectorized IWPP algorithm

```
Input: D:data elements in a multi-dimensional space
Output: D:stable set with all propagations reached
1:  $vec_{shift}$ ← Constant address distance from neighborhood
2:  {Initialization Phase}
3:  …
4:  {Scan Phase}
5:  …
6:  {Wavefront Propagation Phase}
7:  while $currWave.empty() == false$ do
8:    {Identification of elements receiving propagation}
9:    for all $p \in currWave$ do
10:     $vec_p$ ← Extract active elements
11:     $vec_{addr} \leftarrow VecAdd(vec_p, vec_{shift})$
12:     $vec_{neigh} \leftarrow Gather(D, vec_{addr})$
13:     $mask_{cond} \leftarrow VectorPropagationCondition(vec_p, vec_{neigh})$
14:     $vec_{prefixSum} \leftarrow PrefixSum(mask_{cond})$
15:     $nextWave.Insert(vec_{addr}, mask_{cond}, vec_{prefixSum})$
16:   end for
17:   {Propagation}
18:   for all $q \in nextWave$ do
19:     $vec_q$ ← Extract elements
20:     $vec_{addr} \leftarrow VecAdd(vec_q, vec_{shift})$
21:     $vec_{neigh} \leftarrow Gather(D, vec_{addr})$
22:     $D(q) \leftarrow Max/MinReduce(vec_q, vec_{neigh})$
23:   end for
24:   $currWave \leftarrow nextWave$
25:   $nextWave \leftarrow \emptyset$
26: end while
```

Further, vec_p is added to a precomputed vector register (vec_{shift}), which contains the shift values necessary to compute the address of each neighbor element, considering the address of p as a reference (line 11). The shift values will be, for instance, −1 for the left neighbor. The vec_{shift} for an 8-connected neighborhood is shown in Figure 5(A), which uses a Grid of $i \times j$ elements (pixels in an image) to exemplify the effects of the vector instructions. Further, using the addresses values computed for all neighbors, a gather instruction is used to load them into vec_{neigh}, as shown in line 12. In line 13, the propagation condition is computed using p and its neighbors, and the results of this computation will be inserted into a mask vector register whose values identify the cases in which the condition is evaluated true as 1. In order to insert the elements that had the propagation evaluated true in the $nextWave$, we first compute a prefix-sum of the mask register. Each entry of the $vec_{prefixSum}$ vector register will indicate the position, starting from the last element stored into the *nextWave* queue, in which the neighbor should be stored in case it is an information receiver. This insertion process is illustrated in Figure 5(B).

After the elements receiving information are identified, the propagation is computed in lines 18 to 23. We extract the q element from the *nextWave* container (line 19), identify and read its neighbors (lines 20 and 21), and compute the value from the neighborhood that should be propagated to q (line 22). Such as, in the previous stages of the algorithm, multiple q elements may be processed as long as their neighborhoods fit into the vector registers. It is also important to highlight that the same element q could have been inserted multiple times in the *nextWave*, and this could result in having replicas of an element being processed concurrently within a single pass of the vectorized propagation. This will result into a data race during the $D(q)$ update. However, since the same neighborhood is read by each of the replicas of q in the same instruction, the input information will be the same as well as the result for any of the computations. As such, this race condition is classified as a benign[14] race condition and will not influence the operation results.

4.3 | Extending the vectorized IWPP with thread-level parallelism

This section presents our approach to employ thread-level parallelism in IWPP. In this parallelization, the vectorized IWPP algorithm is modified to divide the work by splitting the initial set of wavefront elements identified in the initialization phase among threads. This process is illustrated in Algorithm 5. Multiple domain partitions are created in line 4. This is implemented by creating blocks of consecutive rows with size equals to number of rows in the input domain divided by the number of threads. Further, in lines 6 to 8, the partitions are independently processed by the available threads, which will insert the identified active elements in their wavefront container copies (*currWave*[*tid*]).

Algorithm 5

Vectorized Thread-Level Parallel IWPP algorithm

```
Input: D:data elements in a multi-dimensional space; NT:number of threads used
Output: D:stable set with all propagations reached
1:   $vec_{shift} \leftarrow$ Constant address distance from neighborhood
2:   {Initialization Phase}
3:   …
4:   $D_{partitions} \leftarrow partition(D, NT)$
5:   $tid \leftarrow$ getThreadld()
6:   while $D_i \in D_{partitions}$ do in parallel
7:       $currWave[tid] \leftarrow$ subset active elements from $D_i$
8:   end while
9:   {Wavefront Propagation Phase}
10:  while $currWave[tid].empty() == false$ do in parallel
11:      {Identification of elements receiving propagation}
```

```
12:         for all $p \in currWave[tid]$ do
13:             $vec_p \leftarrow$ Extract active elements
14:             $vec_{addr} \leftarrow VecAdd(vec_p, vec_{shift})$
15:             $vec_{neigh} \leftarrow Gather(D, vec_{addr})$
16:             $mask_{cond} \leftarrow VectorPropagationCondition(vec_p, vec_{neigh})$
17:             $vec_{prefixSum} \leftarrow PrefixSum(mask_{cond})$
18:             $nextWave[tid].Insert(vec_{addr}, mask_{cond}, vec_{prefixSum})$
19:         end for
20:         {Propagation}
21:         barrier
22:         for all $q \in nextWave[tid]$ do
23:             $vec_q \leftarrow$ Extract elements
24:             $vec_{addr} \leftarrow VecAdd(vec_q, vec_{shift})$
25:             $vec_{neigh} \leftarrow Gather(D, vec_{addr})$
26:             $D(q) \leftarrow Max/MinReduce(yec_q, vec_{neigh})$
27:         end for
28:         $currWsve[tid] \leftarrow nextWave[tid]$
29:         $nextWave[tid] \leftarrow \emptyset$
30:     end while
```

The actual propagation phase is presented in lines 10 to 30 of Algorithm 5. This phase is executed in parallel and each thread will be responsible for carrying out propagations of active elements stored in its corresponding container ($currentWave[tid]$). In order to guarantee correctness, we have to assert that all active elements are identified before the propagation is executed. The rest of the algorithm is very similar to the sequential version. However, even though the threads start processing active elements from disjoint partitions of the domain, a propagation may cross the partition. As a consequence, multiple threads may be updating the same domain partition in parallel or even the same data element ($D(q)$) and, as a consequence, a data race may also occur in line 26.

We present this data race in Figure 6(A) using the Morphological Reconstruction operation as an example. As shown in the figure, the computation of the identification phase for both threads will result in inserting e_x in their nextWave. If the neighborhood of e_x read by both threads is the same, they will write the same information in $D(q)$. This results into a benign data race similar to that observed in the vectorization. On the order hand, a more complex scenario occurs when e_x is modified, for instance, by the propagations of another element such as e_{x+2}. In this case, the intermediate result of iteration z may differ depending on the order in which propagations are carried out and threads read the neighborhood. This would lead to one of two cases (C1 and C2) that are presented in Figure 6(A).

The first case (C1) is the more complicated configuration. Assume, for instance, that thread 1 reads neighbors of e_x to the register and, after that thread 2 computes propagations of e_{x+1} and e_x, the neighborhood values read by thread 1 will be outdated and the intermediate results will be as shown in C1. In this configuration, the execution of the next iteration $z + 1$ (Figure 6(B)) would still contain e_{x+1} as an active element, and the propagation resulting from e_{x+1} will change the value of e_x to 8 using the newest neighborhood. The second case (C2) is the one in which both threads read the neighborhood of e_x after it has been updated as a consequence of processing propagations from e_{x+1}. In this case, e_x would be set to its final value in iteration z (Figure 6(C)), and no additional propagation is identified in iteration $z + 1$. In both cases, the result is the same and the data races observed here are benign and could be classified as double checks.[14]

4.4 | Employing the downhill filter approach to increase the parallel efficiency

The original IWPP processes the active elements following a FIFO (First-In, First-Out) order as employed by Vincent.[3] The Downhill filter approach,[15] on the other hand, has shown that the FIFO approach could lead to unnecessary recomputation of propagations, which occurs when an element is modified multiple times due to the interaction among wavefronts. Thus, the Downhill filter approach proposed a propagation order in which elements receiving propagation attain their final value after a single propagation. As such, it is able to eliminate recomputations. This was attained by building a priority queue in which elements with higher values propagate first.

This approach is also interesting in our parallel versions of IWPP because it reduces the possibilities of increasing the recomputation as a consequence of using multiple threads. This increasing occurs in the FIFO approach because as threads start to interact within the data domain, there is a chance that threads will compute propagations using a less updated domain as compared to the sequential version. This problem is worsened as the number of threads increases. In this case, the improvements in the algorithm throughput (elements computed per time unit) due to parallelism may not result in reducing the execution time at the same rate. However, if the Downhill filter is used, elements with higher values will be processed earlier and recomputation will be minimized, benefiting the parallelism efficiency.

In order to take advantage of the ideas proposed in the Downhill filter approach, we have modified the IWPP to allow for the user to select between a FIFO or priority-based container (also called heap) to store active elements. In this case, a regular priority container such as provided by the C++ Standard Template Library (STL)[16] is used.

5 | COOPERATIVE AND OUT-OF-CORE EXECUTION ON HYBRID MACHINES

A significant number of state-of-the-art high-end computing machines are being built using powerful multi-core CPUs and one or more co-processors. In order to fully take advantage of the computing power of these machines, it is important to cooperatively use all available devices. However, the collective use of heterogeneous processors to solve a problem may be a very challenging task. In the case of IWPP, it requires: (i) a parallelization scheme of the IWPP that allows for partitioning the computation without affecting the output results and (ii) a careful division of the computation among the CPU and co-processors. The division of the input image into smaller partitions also allows for the co-processors to be used to execute large images that would not fit entirely in the device memory. The rest of this section first presents an overview of the parallelization approach for IWPP that allows for multiple processors to compute a single image (Section 5.1). Further, in Section 5.2, we discuss the implementation of this parallelization approach.

5.1 | Overview of the parallelization and data partition approach

The cooperative parallel approach we developed divides the input data domain into partitions that are assigned for independent execution in the available devices. The devices carry out local wavefront propagations in parts of the subdomain assigned to them, and a merging phase is executed to assemble the original image and correct border artifacts. In other words, the merging of the problem sub-solutions should consider that a wavefront could have propagated from one partition to another. The propagations among partitions are called inter-partition wavefront propagations, whereas the propagations within a given partition are named here as intra-partition.

Therefore, in the overall solution, the intra-partition and inter-partition steps are executed until at least a single propagation exists between partitions. This asserts that the computation will end only when a stable solution has been reached. Figure 7 shows an example of the processing steps using this strategy and the morphological reconstruction algorithm. First, the input image is divided into two partitions (Figure 7(A)), which are processed independently to compute intra-partition wavefront propagations (Figure 7(B)), inter-partition propagations are computed (Figure 7(C)), and another round of intra-partition computations are executed to generate the final image (Figure 7(D)). In this example, the final result (stability) was obtained with a single inter-partition propagation round, but more iterations may be required in other cases. The same approach can be used in all IWPP algorithms, which is possible because the operations performed by IWPP are commutative and monotonically increasing/decreasing.

5.2 | Cooperative execution and implementation details

5.2.1 | Overall workflow—The cooperative execution is built on the data partition observations and execution scheme presented in the previous section, which shows that an image can be divided for execution as long as a post-processing is performed to correct border artifacts. Therefore, our overall strategy was designed by decomposing the main steps previously described into a workflow of stages shown in Figure 8, which consists of data partition, the computation of propagation within each partition, and inter-partition propagation to allow for information to migrate from partitions.

In this workflow, the image is first divided into multiple disjoint partitions in the “Image Division” stage. This partitioning schemes only creates metadata describing the bounding boxes of the partitions in the original input data buffer. This avoids creating unnecessary copies of the input data domain. Further, an instance of the “Intra-Partition Propagation” stage is instantiated for each input data partition to execute the intra-partition wavefront computations. The “Intra-Partition Propagation” is by far the most expensive stage of the workflow, and its instances can be executed by the CPUs or Intel Phis or both concurrently. Moreover, when all “Intra-Partition Propagation” instances have finished their execution, the “Inter-Partition Propagation” stage is executed to carry out propagations between the partitions of the input data and resolve propagations that start from active element resulting from these inter-partition propagations.

Although it is possible to create multiple copies of the “Inter-Partition Propagation” stage, we have noticed that its computation is very lightweight and, as such, it is more efficiently executed by a single stage instance that executes in the CPU using multiple computing cores. In order to execute the workflow, we developed a workflow manager system that coordinates the stages execution and asserts that dependencies among stage instances are respected.

5.2.2 | Work division—To efficiently utilize multiple computing devices cooperatively, it is necessary to minimize the load imbalance between “Intra-Partition Propagation” stage instances assigned to different processors. In our implementation, this is addressed in the “Image Division” stage that creates partitions of the original input domain with sizes that are proportional to the computing power of the devices. This division considers the entire CPU with its the multiple cores as a single device (CPU_m), and each of the j Intel Phis as another device (Phi_j). Further, using the performance of the single core execution as a baseline, we compute speedups of the ($S_{CPU_m} = Time_{CPU_{sequential}} / Time_{CPU_m}$) and of each j-th Intel Phi ($S_{Phi_j} = Time_{CPU_{sequential}} / Time_{Phi_j}$). These speedup calculations are obtained in profiling phase carried out before the actual execution of the cooperative runs and should be provided by the user.

Given the speedups calculated for all processors, we can estimate the proportion of the machine computing power that is delivered by each processor. This value is used to calculate the ratio of the image domain (partition size) that will be assigned to each device. For instance, if a CPU and one or more Intel Phi co-processors are used, the ratio of the input data domain assigned to the CPU is $R_{CPU_m} = S_{CPU_m} / (S_{CPU_m} + \sum_j S_{Phi_j})$, whereas the j-th Intel Phi will compute a ratio $R_{Phi_j} = S_{Phi_j} / (S_{CPU_m} + \sum_j S_{Phi_j})$. The actual computation workflow is dispatched for execution with our Task Execution Engine (TEE - see Figure 8), which will guarantee that dependencies are resolved before the actual execution of stage instances (tasks). It also assigns one computing thread per computing device, and this thread will communicate with the TEE scheduler to receive partitions that should be processed in that device.

5.2.3 | Discussion and limitations—There are important decisions that we had to take during the development of the partitioning scheme that may substantially affect its performance. For instance, we have considered the CPU as a single device instead of considering each computing core as a device. The main reason for using this strategy is that a smaller number of partitions is created in this approach (one per CPU instead of one per CPU-core) and, as a consequence, it reduces the inter-partition propagations.

We have also created a single partition per processor as our default strategy instead of creating multiple partitions. This decision has been made based on the observation that the largest the partition is, the better tends to be the processors efficiency. This approach, however, is modified if the entire partition or image sub-domain assigned to a co-processor does not fit in its memory. In this case, we will create smaller partitions to allow execution of large images, and these smaller partitions are computed recursively by dividing partitions into half of their sizes until they do not fit in the co-processor memory. As one may have noticed, in the current version of our solution, we assume that the memory of the CPU will be sufficient to store the entire input domain/image. As a future work, we intend to remove this limitation with a hierarchical execution scheme in which other storage layers could also be used.

6 | EXPERIMENTAL EVALUATION

The experimental evaluation was carried out using the Morphological Reconstruction and Fill Holes algorithms and the images collected and used in our early brain tumor research initiatives.[17] The experimental results were repeated 10 times and a variance not higher than 1% was observed. We have also employed four machines: The first machine is configured with two 8-core Intel Xeon E5-processors with 2.6 GHz, 64 GB of RAM, and one Intel Xeon Phi 7120P. The second machine hosts two 8-core Intel Xeon E5-processors with 2.7 GHz, 20 MB of L3, 32 GB of RAM, and two Intel Xeon Phi SE10P. The third machine is equipped with a 8-core Intel Xeon E5-processor with 2.7 GHz, 20 MB of L3, 32 GB of RAM, one GPU Nvidia Tesla K20. The last machine contains an Intel Phi 7250 which is used as a standalone (or bootable) processor. The first machine is used for most of the experiments, whereas the second machine is employed in experiments involving the cooperative execution. The third machine, on the other hand, was used to measure and compare the performance of the target operations in GPU devices. The input images vary in size and tissue coverage. Tissue coverage refers to the percentage of the image area covered with tissue (see Figure 9 for examples of different tissue coverage).

In order to evaluate the performance of the proposed methods, we have compared the implementations to efficient CPU[3] and GPU[1,18] implementations of the IWPP.We have also benchmarked the processors using the STREAM benchmark[19] to compute regular memory access bandwidth. The memory bandwidth with these benchmarks are presented in Table 3. The processors were benchmarked with the intention of collecting additional information used to explain the performance of the target operations.

6.1 | Impact of vectorization

This section evaluates the performance gains of the proposed vectorized IWPP as compared to the original non-vectorized version. The experiments were carried out using the Intel Phi SE10P and 4K×4K input images with different coverage characteristics, as illustrated in Figure 9. In addition, we have employed the IWPP with a regular FIFO queue structure to store the wavefront elements and have evaluated the Morphological Reconstruction and Fill Holes operations.

As presented in Figure 10, both operations have significantly benefited from the vectorized version of IWPP. The speedups attained on top of the non-vectorized versions of the operations are up to 5.17× and 5.63×, respectively, for the Morphological Reconstruction and Fill Holes. Although the gains are high, they are not linear with respect to the number of data elements that fit into a vector register of the Intel Phi (16). A smaller gain was expected because of the larger number of instructions executed by the vectorized version as compared to the non-vectorized. For instance, the vectorized version requires an additional instruction to load the input data elements to a vector register and a more complex strategy to compute the position of elements into the queue (prefix-sum).

6.2 | Performance of operations for different input sizes and tissue coverage

These experiments evaluate the impact of the input image size and tissue coverage to the overall performance of the operations on the Intel Phi SE10P as compared to the sequential execution on the CPU. This evaluation presents and discusses the performance of the Morphological Reconstruction with the FIFO queue only because the Fill Holes results are very similar. The results are presented in Figure 11. As shown, the performance gains of the Intel Phi parallelization are significant for all configurations, but the improvements increase for images with larger coverage and size. This occurs because larger coverage and image sizes lead to more computations, which, in turn, better offsets the initialization overheads and exploits the parallelism of the co-processor. However, it is important to notice that using images bigger than 8K×8K has not led to a substantial performance improvement.

6.3 | Comparison of Intel Phi-based vectorized IWPP to efficient CPU and GPU versions

This section presents the performance of our proposed approach on the Intel Phi SE10P, 7120P, and 7250 as compared to other efficient implementations for multi-core CPUs and GPUs.[1] The multi-core CPU version was executed on Intel E5-processors with 2.6 GHz and employed the 16 computing cores available. The SE10P and 7120P devices are equipped with 61 cores, but the latter has a higher clock rate. The GPU used is the NVIDIA K20. Details about the co-processors are presented in Table 3. We have also compared different strategies for processing the wavefront elements: (i) FIFO that processes elements in the order in which they are inserted into the wavefront and (ii) Priority that uses the Downhill strategy to first compute elements with higher intensity values.

The performance of all versions of IWPP for the Morphological Reconstruction operation in the processors evaluated is presented in Figure 12. As shown, the multi-core CPU version was able to scale well as it attained speedups of about 14× on top of the sequential version. Further, as we compare the different Intel Phi devices, it was observed that the 7120P improved the performance of the operations in about 1.14× as compared to the SE10P when using either the FIFO or Priority (Downhill) approaches. This improvement is consistent with the differences in the computation capabilities of the devices. Further, the 7250 Intel Phi is about 2× faster than the 7120P. This improvement is mainly due to the much higher memory bandwidth of the 7250 (Table 3). The GPU version attained a slightly better performance than the 7120P Intel Phi with FIFO, but the use of the Priority container leads the 7120P to a performance that is about 1.6× superior than the GPU. The improvements with the Downhill approach are a consequence of processing less elements in the wavefront as compared to the execution using the FIFO strategy. It is worth noting that we have implemented IWPP for GPUs using an efficient priority queue proposed by He et al,[20] but it did not lead to performance improvements on top of the FIFO case because of the higher costs of executing/managing this structure in the GPU. Finally, the 7250 has achieved higher performance than the K20 GPU for all configurations. It is 6.21× faster than the multi-core CPU configuration for 16K × 16K images.

6.4 | Effects of cooperative execution

These experiments look at the performance of cooperative execution on CPU and Intel Xeon Phi and use input images with 24K × 24K and 32K × 32K pixels. The experiments were executed on our computing setting equipped with two 8-core Intel Xeon E5-processors with 2.7 GHz and two Intel Xeon Phi SE10P co-processors. This configuration provides a more appropriate environment for the cooperative execution evaluation. This machine is also equipped with a faster CPU as compared to the one used in previous sections.

We evaluated five versions of the Morphological Reconstruction: (i) the single core sequential CPU version, (ii) the CPU multithreaded version, (iii) the version with cooperative execution that uses one Intel Phi and the multiple CPU cores available, (iv) the cooperative version that employs two Intel Phi co-processors, and (v) the cooperative version with two Intel Phi co-processors and the CPU cores. One CPU core was reserved for managing each Intel Xeon Phi co-processor. The results reported include all overheads, for instance, the data transfer times between the CPU and the co-processors, the division of the input data domain, and the correction of border effects.

First, in Table 4, we present the amount of work assigned to each processor type in cooperative execution using heterogeneous processors. The partitioning strategy, as described in Section 5, considered the computing power of each of the processors to compute the work division. The performance results are presented in Figure 13. The combined use of co-processors significantly improved performance in all settings. The execution using two co-processors achieved speedups of about 1.78 ×(32K × 32K image) as compared to runs with a single co-processor. The main limitation for attaining higher performance gains is the growth in the data transfer costs between the CPU and the co-processors when multiple devices are used. In this case, the co-processors try to access and transfer data concurrently using the same communication channels,[21] which results in a contention that increases in 1.54× the data transfer times. The performance improvements with the use of CPU and one or two co-processors are proportional to the amount of computation assigned to the CPU in both cases. This shows that the data partition strategy was effective in assigning computation to the devices.

7 | RELATED WORK

The IWPP resembles graph scan algorithms with multiple sources, which have been the target of a number of recent research projects that implemented, for instance, Breadth-First Search (BFS).[22,23] Hong et al[22] presented approaches to minimize the load imbalance that occurs when processing graphs in which the number of edges may vary from vertices. Tao et al[23] described the acceleration of BFS in the Intel Phi with techniques to use SIMD instructions in this operation. However, their work uses atomic instructions in some phases of the algorithm, ie, the expansion of nodes, which poses as a limiting factor to attain maximum performance in the Intel Phi. Although efficient in their scenarios, both optimizations would have little impact to improve IWPP because: (i) the graph in which the information propagates in IWPP is composed of image pixels and the neighborhood is balanced (size of the structuring element used); and (ii) the use of atomic instructions limits the gains with vectorization in the IWPP.

The Morphological Reconstruction and Fill Holes were defined in the work of Vincent,[24] which described the efficiency of using a queue (FIFO) queue to track active elements. A parallel CPU cluster-based version of Morphological Reconstruction was developed in the work of Laurent and Roman,[25] while other works used devices such as Field-Programmable Gate Arrays (FPGAs) and GPUs to implement this operation.[26–28] A common limitation with these solutions is that they were not built on top of the must efficient sequential algorithm that uses queues. Instead, they use less efficient versions of the algorithm that repeatedly scan the entire image domain, for instance, using raster and anti-raster strategies until at least a single pixel/element propagates during a pass. This strategy tends to be inefficient because, even if a few pixels are active, the scan will unnecessarily touch all elements in the domain.

The approach proposed on this work, on the other hand, is built on top of the most efficient versions of Morphological Reconstruction and Fill Holes. The first GPU accelerated implementation of the efficient version these algorithms (using queues) was presented in the works of Teodoro et al,[1,18] whereas in other work of Teodoro et al,[7] authors presented an initial non-vectorized version of the IWPP on the Intel Phi. Both works have been used as baselines in our performance evaluation in which the vectorized IWPP proposed in this work attains better performance.

The use of hybrid machines equipped with accelerators is an increasingly important topic.[29–35] At the same time that these machines offer several opportunities for accelerating general purpose applications, they also create new challenges with respect to their efficient use. Several previous works have developed run-time systems[29,33–35] that simplify the use of GPU and CPUs by providing strategies to schedule operations and by optimizing data transfers among these devices. Other strategies, on the other hand, tried to develop domain specific run-time systems, for instance, for application that perform generalized reductions[31] or Stencil computations.[36] In our work, we also built a framework for efficient execution of a domain specific operation (IWPP). However, the division of work in IWPP required a complex approach that is implied in the redesign of the algorithm, whereas most of the previous works deal with operations in which there are several fine-grain and independent tasks that are partitioned for parallel execution in the available processors.[29,31,35,37]

8 | CONCLUSIONS AND FUTURE DIRECTIONS

The use of WSIs in clinical and research studies can reveal interesting cellular and sub-cellular characteristics of several disease mechanisms, for instance, in cancer. However informative, these datasets of images are very large and the computation of a single image can take hours in a regular desktop machine, and a medium scale study would include processing hundreds of WSIs. This high computation demand requires the use of parallel machines to accelerate image analysis workflows and enable a quick processing and inspection of these datasets.

Therefore, in this paper, we have proposed and implemented an efficient version of IWPP for the Intel Phi co-processor and hybrid machines. The IWPP is a common computation structure found in several image analysis operations typically employed in image segmentation, whose optimization can significantly improve the performance of our motivating application that processes WSIs. This new IWPP presented several optimizations that include: a redesigned algorithmic solution that employs vectorization, the use of thread-level parallelism, the evaluation of different strategies to process wavefront elements, and cooperative execution on hybrid machines equipped with CPUs and Intel Phi co-processors. The optimizations have been thoroughly evaluated to show, for instance, that the vectorized version of IWPP proposed in this paper attains a performance gain of up to about 5.6× on top of the original algorithm on the Intel Phi. We have also compared our propositions to the most efficient implementation on CPUs and GPUs, which shows that our approach is about 3.14× more efficient than the GPU based solution.[1] The cooperative execution has demonstrated to be efficient in using multiple processors together. In an execution using two Intel Phi co-processors and a multi-core CPU, we have attained a speedup of up to 4.86× on top of the multi-core CPU version of Morphological Reconstruction. This level of performance is a key aspect to allow for the computation of large datasets of WSIs as required in real-world studies.

In the future work, we intend to improve the performance of our IWPP execution framework by extending and deploying our solutions in a cluster environment. This work will involve looking into the data domain partition hierarchically, such that a domain partitioned for computation in multiple nodes of the distributed environment would have to be repartitioned for cooperative execution within each machine equipped with multiple processors. This approach will also consider that data exchange among nodes should be minimized because of the higher costs involved. We envision to address this problem with different tile padding strategies, which could include using overlapping tiles with overlap sizes that vary according to the cost of communication among nodes in the system. This leads to an interesting trade-off of reducing the communication demands as the overlap increases with the cost of processing overlapping areas multiple times.

Acknowledgments

This work was supported in part by 1U24CA180924-01A1 from the National Cancer Institute (NCI), R01LM011119-01 and R01LM009239 from the U.S. National Library of Medicine (NLM), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), and National Institutes of Health (NIH) K25CA181503. This research used resources provided by the XSEDE Science Gateways program under grant TG-ASC130023.

Funding information

National Cancer Institute (NCI), Grant/Award Number: 1U24CA180924-01A1; U.S. National Library of Medicine (NLM), Grant/Award Number: R01LM011119-01, and R01LM009239; Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq); Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES); National Institutes of Health (NIH), Grant/Award Number: K25CA181503; XSEDE Science Gateways, Grant/Award Number: TG-ASC130023

References

1. Teodoro G, Pan T, Kurc T, Kong J, Cooper L, Saltz J. Efficient irregular wavefront propagation algorithms on hybrid CPU-GPU machines. Parallel Comput. 2013; 39(4–5):189–211. [PubMed: 23908562]

2. Kong J, Cooper LA, Wang F, et al. Machine-based morphologic analysis of glioblastoma using whole-slide pathology images uncovers clinically relevant molecular correlates. PLOS One. 2013; 8(11):e81049. [PubMed: 24236209]
3. Vincent Luc. Morphological grayscale reconstruction in image analysis: applications and efficient algorithms. IEEE Trans Image Process. 1993; 2(2):176–201. [PubMed: 18296207]
4. Soille Pierre. Morphological Image Analysis: Principles and Applications. 2nd. Secaucus, NJ: Springer-Verlag New York, Inc; 2003.
5. Vincent L, Soille P. Watersheds in digital spaces an efficient algorithm based on immersion simulations. IEEE Trans Pattern Anal Mach Intell. 1991; 13(6):583–598.
6. Vincent L. Paper presented at: IEEE International Conference on Computer Vision and Pattern Recognition. Maui, HI: 1991. Exact Euclidean distance function by chain propagations.
7. Teodoro G, Kurc T, Kong J, Cooper L, Saltz J. Paper presented at: IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS). Phoenix, AZ: 2014. Comparative performance analysis of Intel (R) Xeon Phi (TM), GPU, and CPU: a case study from microscopy image analysis.
8. Gomes JM, Teodoro G, de Melo A, Kong J, Kurc T, Saltz JH. Paper presented at: 2015 27th International Symposiumon Computer Architecture and High Performance Computing (SBAC-PAD). Florianopolis, Brazil: 2015. Efficient irregular wavefront propagation algorithms on Intel (R) Xeon Phi (TM).
9. Cooper L, Kong J, Gutman D, et al. An integrative approach for in silico glioma research. IEEE Trans Biomed Eng. 2010; 57(10):2617–2621. [PubMed: 20656651]
10. Teodoro G, Pan T, Kurc T, et al. Region templates: data representation and management for high-throughput image analysis. Parallel Comput. 2014; 40(10):589–610. [PubMed: 26139953]
11. Ruifrok AC, Johnston DA. Quantification of histochemical staining by color deconvolution. Anal Quant Cytol Histol. 2001; 23(4):291–299. [PubMed: 11531144]
12. Jeffers J, Reinders J. Intel Xeon Phi Coprocessor High-Performance Programming. Boston, MA: Elsevier Waltham; 2013.
13. Gonzalez RC, Woods RE, Eddins SL. Digital Image Processing Using MATLAB. Natick, MA: MathWorks; 2010. Morphological reconstruction.
14. Narayanasamy S, Wang Z, Tigani J, Edwards A, Calder B. Paper presented at: Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation. San Diego, CA: 2007. Automatically classifying benign and harmful data races using replay analysis.
15. Robinson K, Whelan PF. Efficient morphological reconstruction: a downhill filter. Pattern Recogn Lett. 2004; 25(15):1759–1767.
16. Musser DR, Derge GJ, Saini A. STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library. 2nd. Boston, MA: Addison-Wesley, Longman Publishing Co, Inc; 2001.
17. Saltz JH, Kurc T, Cholleti S. , et al. Paper presented at: Annual Symposium of American Medical Informatics Association 2010 Summit on Translational Bioinformatics (AMIA-TBI). Washington, DC: 2010. Multi-scale, integrative study of brain tumor: In silico brain tumor research center.
18. Teodoro G, Pan T, Kurc TM, Kong J, Cooper L, Saltz JH. A Fast Parallel Implementation of Queue-Based Morphological Reconstruction Using GPUs. Atlanta, GA: Emory University; 2012.
19. McCalpin JD. Memory bandwidth and machine balance in current high performance computers. IEEE Computer Society Technical Committee on Computer Architecture Newsletter. 1995:19–25.
20. He X, Agarwa LD, Prasad SK. Paper presented at: 19th International Conference on High Performance Computing (HiPC). Pune, India: 2012. Design and implementation of a parallel priority queue on many-core architectures.
21. Newburn CJ, Dmitriev S, Narayanaswamy R. , et al. Paper presented at: IEEE 27th International, Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW). Cambridge, MA: 2013. Offload compiler runtime for the Intel (R) Xeon Phi coprocessor.
22. Hong S, Kim SK, Oguntebi T, Olukotun K. Paper presented at: Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming. San Antonio, TX: 2011. Accelerating CUDA graph algorithms at maximum warp.

23. Tao G, Yutong L, Guang S. Paper presented at: 2013 IEEE 27th International on Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW). Cambridge, MA: 2013. Using MIC to accelerate a typical data-intensive application: the breadth-first search.

24. Vincent L. Mathematical Morphology in Image Processing. Optical engineering New York, NY: Marcel-Dekker; 1992. Morphological algorithms.

25. Laurent C, Roman J. Paper presented at: Third International Conference on Vector and Parallel Processing, VECPAR ‘98. London, UK: 1999. Parallel implementation of morphological connected operators based on irregular data structures.

26. Karas P. Efficient Computation of Morphological Greyscale Reconstruction. In: Matyska L, Kozubek M, Vojnar T, Zemcík P, Antos D, editorsSixth Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS’10) – Selected Papers. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik; 2011. 54–61. OpenAccess Series in Informatics (OASIcs). Vol. 16. MEMICS; 2010. 54

27. Jivet I, Brindusescu A, Bogdanov I. Image contrast enhancement using morphological decomposition by reconstruction. WSEAS Trans Cir Sys. 2008; 7(8):822–831.

28. Anacona-Mosquera O, Vinhal G, Sampaio RC, Teodoro G, Jacobi RP, Llanos CH. Paper presented at: 30th Symposium on Integrated Circuits and Systems Design (SBCCI). Fortaleza, Brazil: 2017. Efficient hardware implementation of morphological reconstruction based on sequential reconstruction algorithm.

29. Luk CK, Hong S, Kim H. Paper presented at: 42nd International Symposium on Microarchitecture (MICRO). New York, NY: 2009. Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping.

30. Augonnet C, Thibault S, Namyst R, Wacrenier PA. Paper presented at: 15th International Euro-Par Conference on Parallel Processing. Delft, The Netherlands: 2009. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures.

31. Ravi VT, Ma W, Chiu D, Agrawal G. Paper presented at: Proceedings of the 24th ACM International Conference on Supercomputing. Tsukuba, Japan: 2010. Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations.

32. Huo X, Ravi VT, Agrawal G. Paper presented at: 18th International Conference on High Performance Computing (HiPC). Bangalore, India: 2011. Porting irregular reductions on heterogeneous CPU-GPU configurations.

33. Bueno J, Planas J, Duran A. , et al. Paper presented at: IEEE 26th International Parallel Distributed Processing Symposium (IPDPS). Shanghai, China: 2012. Productive programming of GPU clusters with OmpSs.

34. Rossbach CJ, Currey J, Silberstein M, Ray B, Witchel E. Paper presented at: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP ‘11. Cascais, Portugal: 2011. PTask: Operating system abstractions to manage GPUs as compute devices.

35. Gautier T, Lima JVF, Maillard N, Raffin B. Paper presented at: 2013 IEEE International Symposium on Parallel and Distributed Processing. Boston, MA: 2013. Xkaapi: A runtime system for data-flow task programming on heterogeneous architectures.

36. Holewinski J, Pouchet LN, Sadayappan P. Paper presented at: Proceedings of the 26th ACM International Conference on Supercomputing, ICS ‘12; 2012. Venice, Italy: High-performance code generation for stencil computations on GPU architectures.

37. Augonnet C, Aumage O, Furmento N, Namyst R, Thibault S. StarPU-MPI: task programming over clusters of machines enhanced with accelerators. In: Träff JL, Benkner S, Dongarra J, editorsRecent Advances in the Message Passing Interface: The 19th European MPI Users’ Group Meeting (EuroMPI 2012). Vol. 7490. Vienna, Autria: Springer; 2012. Lecture Notes in Computer Science

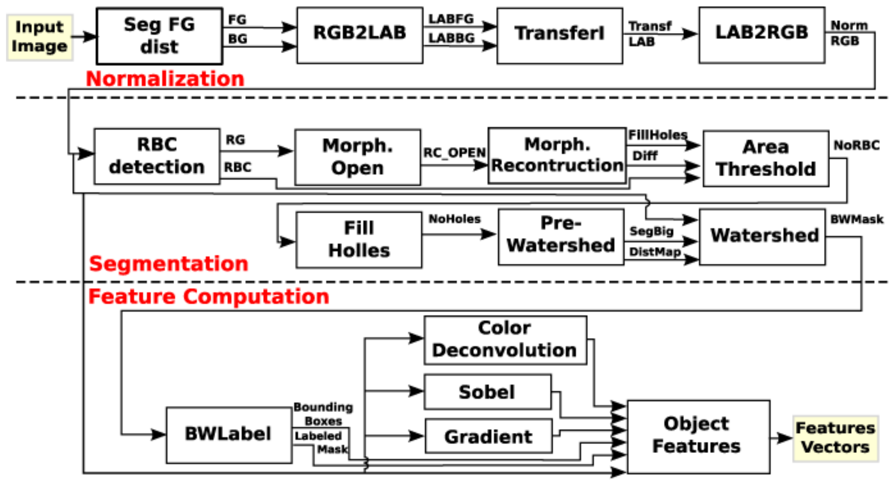


FIGURE 1.

Microscopy Image Analysis Workflow with normalization, segmentation and feature computation stages and their internal fine-grain operations

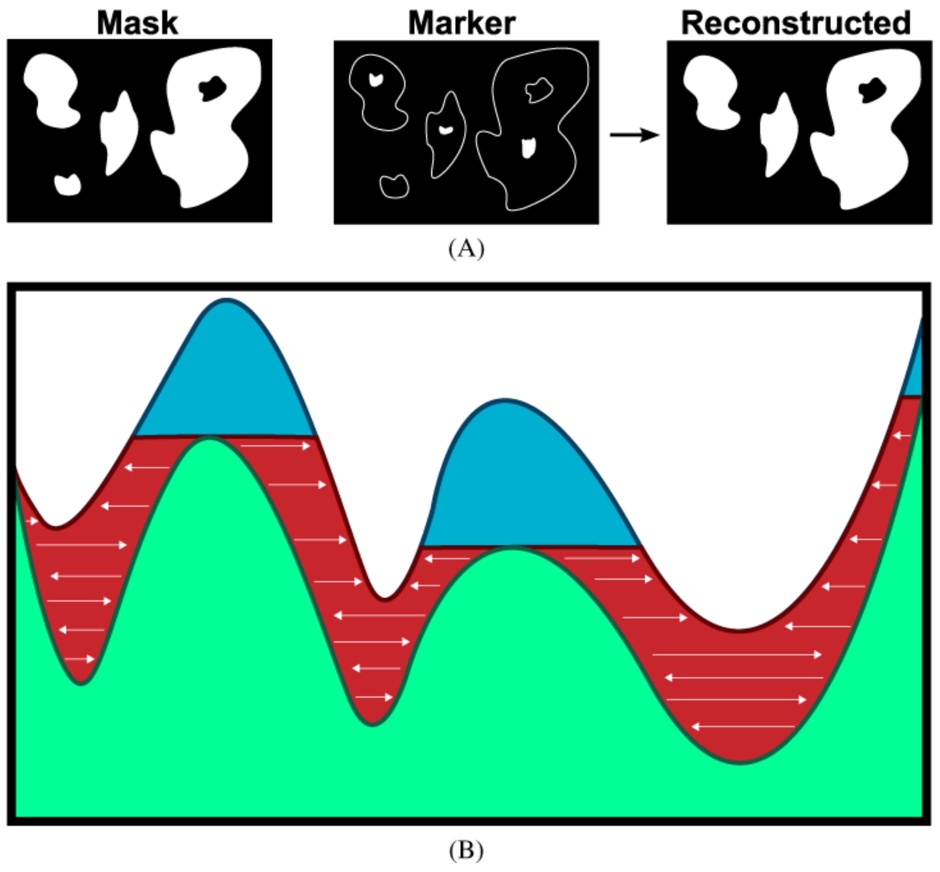


FIGURE 2.
Examples of morphological reconstruction in binary and grayscale images. A, Binary morphological reconstruction. The mask image limits the propagations starting in patches from the marker image; B, Grayscale morphological reconstruction in 1 dimension. The marker image intensity profile is represented as the green line, and the mask image intensity profile is represented as the blue line. The final image intensity profile after reconstruction is the red line. The arrows show the directions of propagations from the marker intensity profile to the mask intensity profile. The red region represents changes introduced by the morphological reconstruction

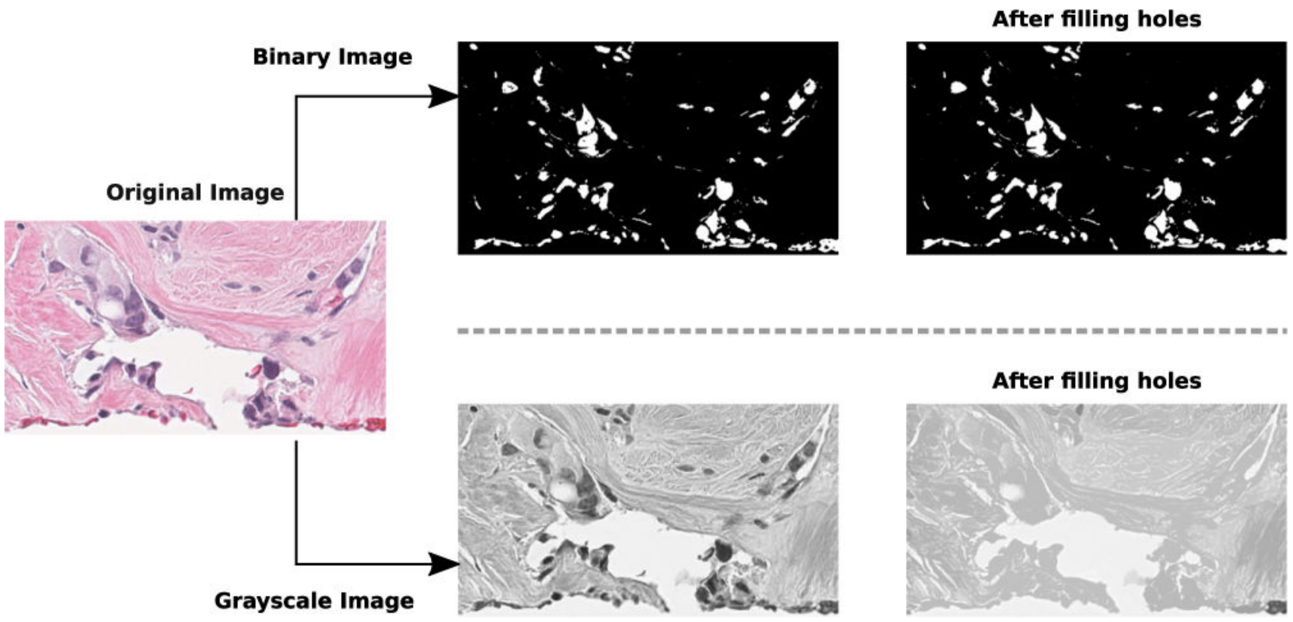


FIGURE 3.
Fill holes in a binary and in a grayscale image. Connected components (those with surrounding border) are filled and noise is reduced

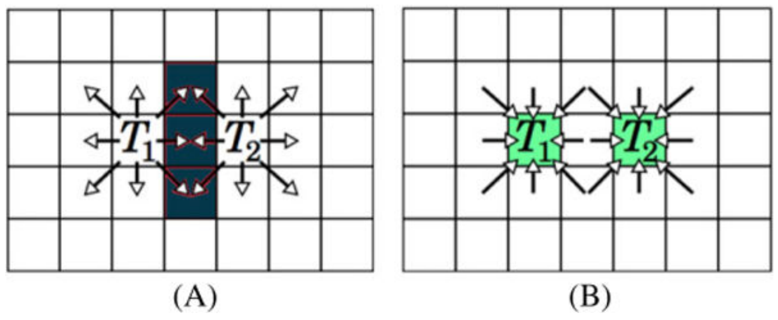
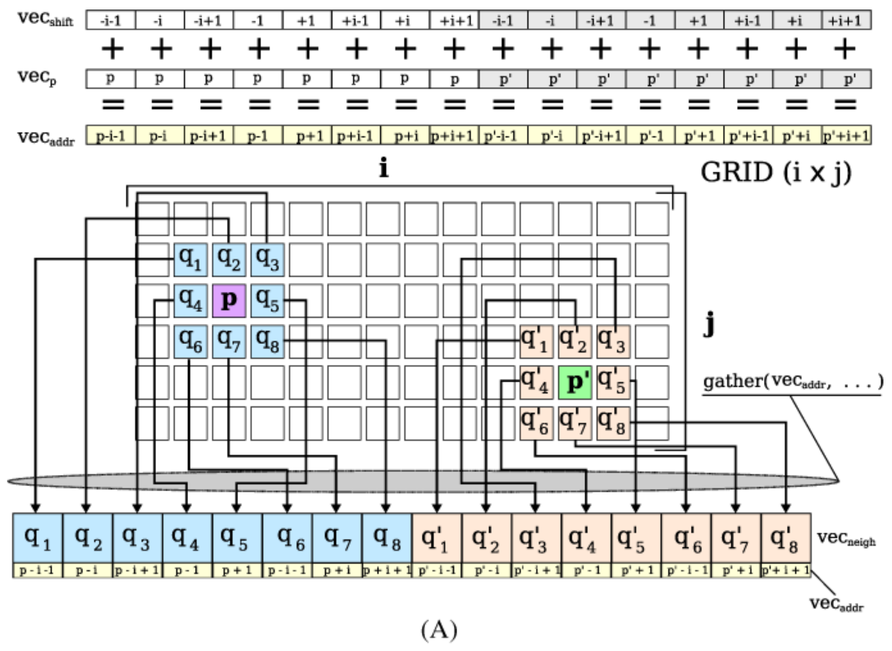
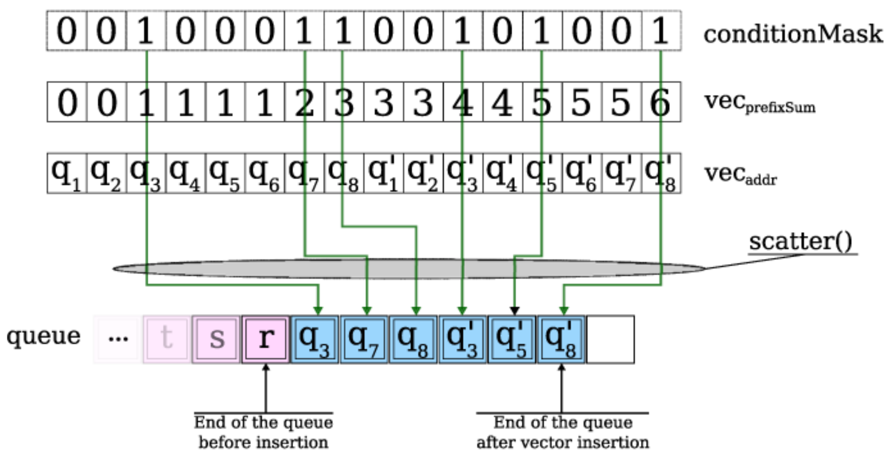


FIGURE 4.

Example race condition in the original IWPP algorithm A, when active elements (T_1 and T_2) propagate their values to a common element, and; B, inversion of the propagation direction with the new IWPP: elements receiving (T_1 and T_2) information become active and collect the information from their neighbors


vec_shift
-i-1 -i -i+1 -1 +1 +i-1 +i +i+1 -i-1 -i -i+1 -1 +1 +i-1 +i +i+1
vec_p
p p p p p p p p p' p' p' p' p' p' p' p'
vec_addr
p-i-1 p-i p-i+1 p-1 p+1 p+i-1 p+i p+i+1 p'-i-1 p'-i p'-i+1 p'-1 p'+1 p'+i-1 p'+i p'+i+1
i
GRID (i x j)
j
gather(vec_addr, . . .)
q1 q2 q3 q4 p q5 q6 q7 q8
q'1 q'2 q'3 q'4 p' q'5 q'6 q'7 q'8
vec_neigh
p-i-1 p-i p-i+1 p-1 p+1 p-i-1 p+i p+i+1 p'-i-1 p'-i p'-i+1 p'-1 p'+1 p'-i-1 p'+i p'+i+1
vec_addr

(A)


0 0 1 0 0 0 1 1 0 0 1 0 1 0 0 1
conditionMask
0 0 1 1 1 1 2 3 3 3 4 4 5 5 5 6
vec_prefixSum
q1 q2 q3 q4 q5 q6 q7 q8 q'1 q'2 q'3 q'4 q'5 q'6 q'7 q'8
vec_addr
scatter()
queue
... t s r q3 q7 q8 q'3 q'5 q'8
End of the queue before insertion
End of the queue after vector insertion

(B)

FIGURE 5.

Illustration of the vectorized steps of the IWPP in a Grid with i×j pixel elements. The part A, presents the process of loading 8-connected neighborhood for two elements p and p'; B, refers to the insertion of elements receiving information into the *nextWave* queue container

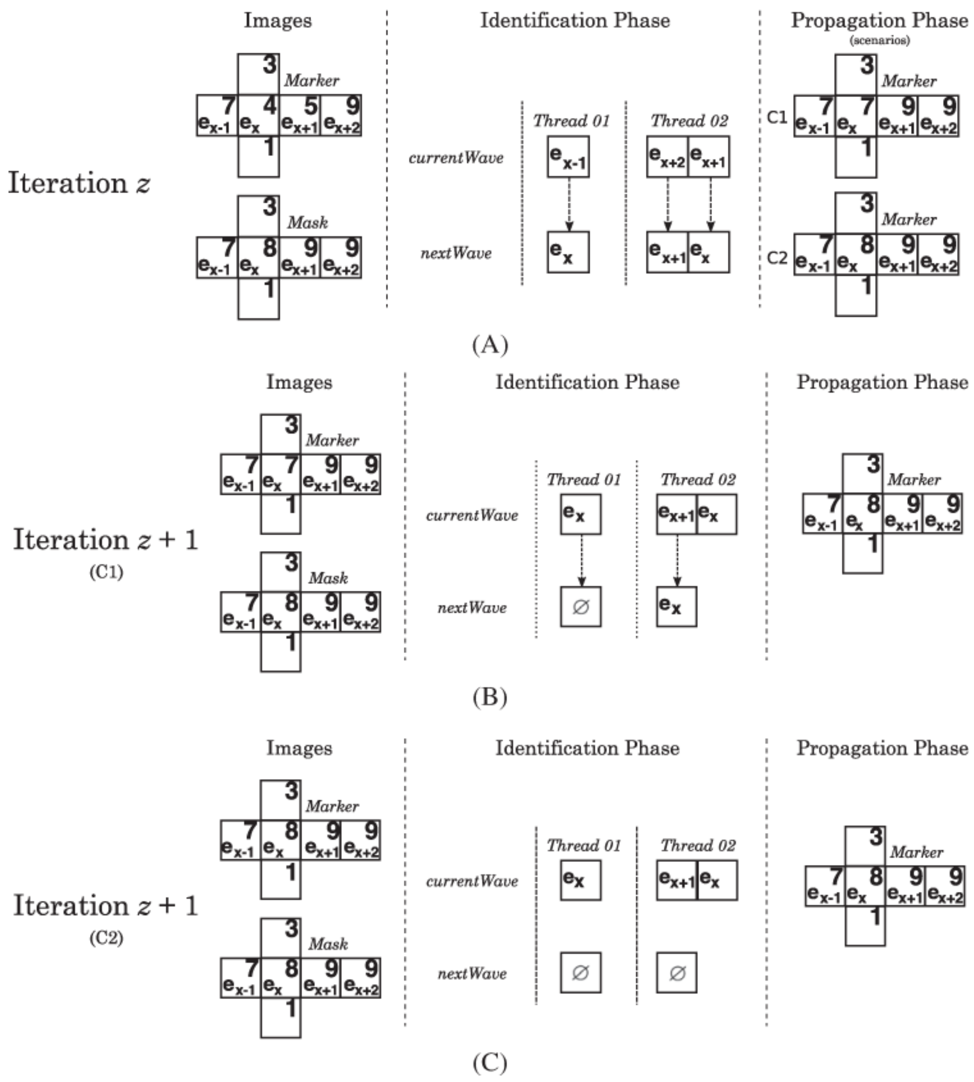


FIGURE 6.
Benign data race for Morphological Reconstruction algorithm. The computation of an image may lead to different intermediate results C1 and C2 in iteration *z* of the IWPP. However, the computation in iteration *z* + 1 will achieve the same output results. A, Computation for the input in iteration *z* may lead to scenarios C1 and C2 depending in the order in which Threads 01 and 02 process their wavefront element; B, Scenario C1 during iteration *z* + 1; C, Scenario C2 during iteration *z* + 1

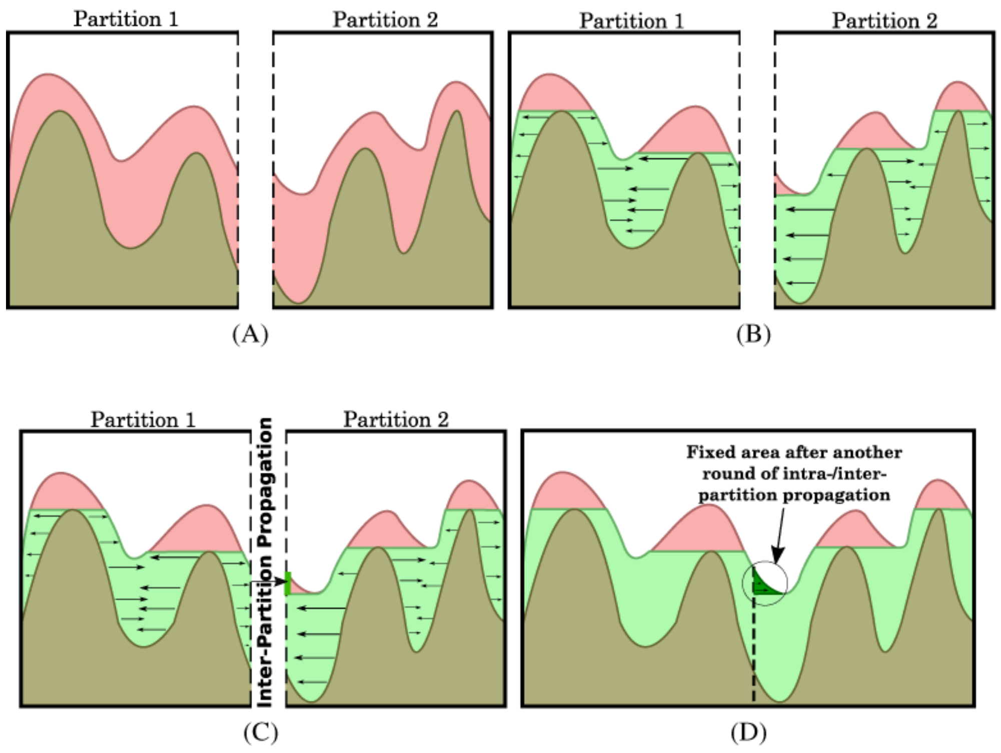

Partition 1
Partition 2
Partition 1
Partition 2
(A)
(B)
Partition 1
Inter-Partition Propagation
Partition 2
Fixed area after another round of intra-/inter-partition propagation
(C)
(D)

FIGURE 7.

IWPP decomposition of the image input domain for parallel execution using multiple devices. The image is divided into sub-domains/partitions that are processed independently. The partitions are then merged and inter-partition propagations are computed to generate the correct output results. A, Input image divided into two partitions; B, Results after intra-partitions computed independently; C, Inter-partition wave propagations; D, Output image after changes resulting from inter-partition wave propagations.

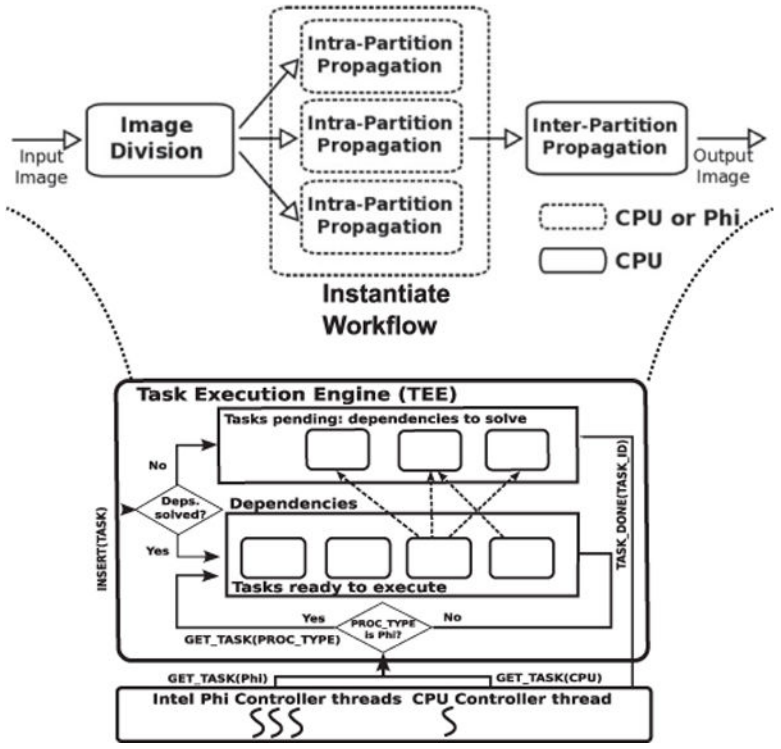


FIGURE 8.
Computation workflow for the IWPP execution on multiple computing devices

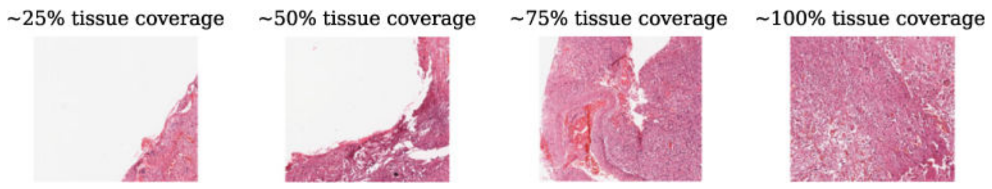


FIGURE 9.
Raw images with different tissue coverage.

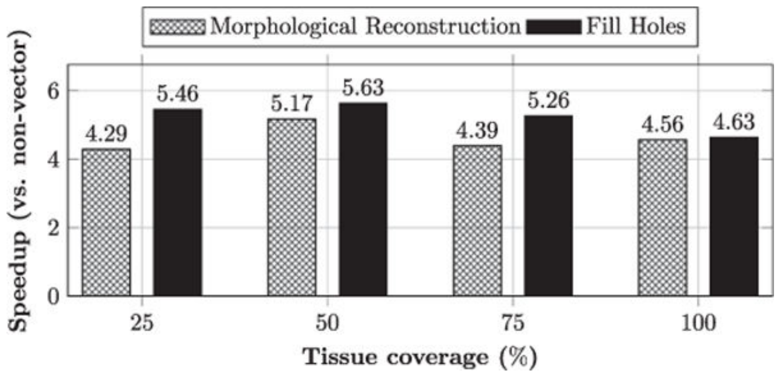


FIGURE 10.

Speedups attained by the proposed vectorized IWPP as compared to the original non-vectorized approach as the image tissue coverage is varied

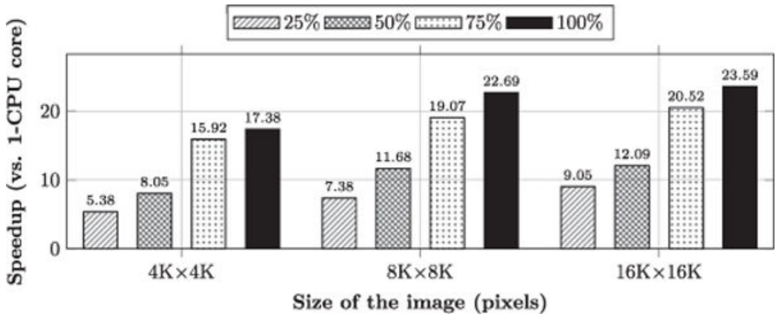


FIGURE 11.
Speedups of the Morphological Reconstruction on the Intel Phi SE10P vs the sequential CPU execution as the input image size and tissue coverage are varied

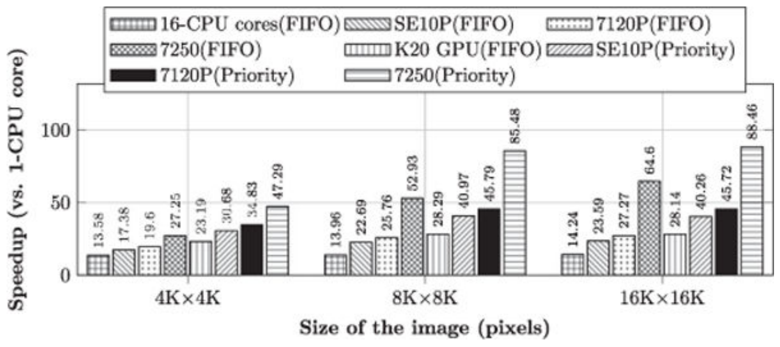


FIGURE 12.

Comparative performance of IWPP on multiple processors using the Morphological Reconstruction and images with 100% coverage. The speedups are computed using the sequential version of the operation as a reference.

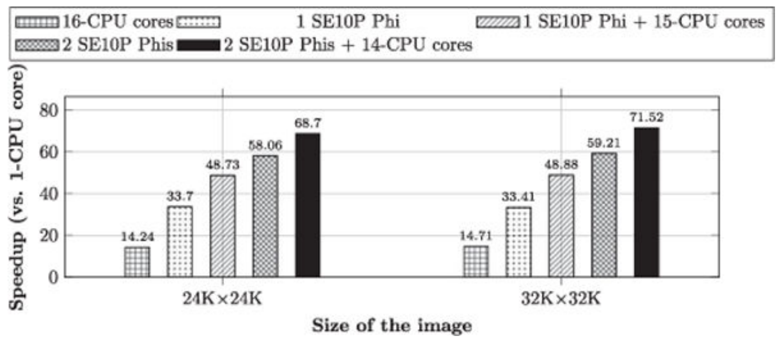


FIGURE 13.

Evaluation of Cooperative Execution on Heterogeneous Processors. A CPU core is reserved to manage the co-processors whenever it is used

TABLE 1

Operations used in each of the application main stages and their parallelism strategy

Operations	Description	Parallelism
Normalization Phase		
Seg FG dist	Segment foreground from background w/discriminant functions	Data
RGB2LAB	Convert from RGB to LAB	Data
TransferI	Map color distribution of an image to that of the target image	Data
LAB2RGB	Convert from LAB to RGB	Data
Segmentation Phase		
Covert RGB to grayscale	Covert a RGB image into a grayscale image	Data
Morphological Open	Opening removes small objects and fills holes in foreground	Data
Morphological Reconstruction	Flood-fill a marker image that is limited by a mask image	IWPP
Area Threshold	Remove objects outside an area range	IWPP and Reduction
Fill Holes	Fill holes objects w/a flood-fill starting at selected points	IWPP
Distance Transform	Compute min distance from foreground pixels to background	IWPP
Watershed	Separate overlapping objects	IWPP
Feature Computation Phase		
BWLabel	Label components (objects) of a mask image with the same value	IWPP
Color Deconvolution[11]	Separate multi-stained biological images in different channels	Data
Gradient	Compute image gradient in x, y	Data
Sobel Edge	Compute Sobel Edge	Data
Object Features	Compute statistics (mean, median, max, etc) for each object	Object

TABLE 2

Characteristics of the Intel Phi processors used

Processor	Name	Cores	Freq.
SE10P	KNC	61	1.10 GHz
7120P	KNC	61	1.33 GHz
7250	KNL	68	1.60 GHz

TABLE 3

Processors characteristics

	K20 GPU	**SE10P**	**7120P**	**7250**
Number of cores	2496	61	61	68
Processor core clock (MHz)	706	1100	1330	1600
Bandwidth - Regular access (GB/s)	148	160	177	460

TABLE 4

Ratio of the input data domain assigned to each processor type

Devices	**1 Phi + CPU (*multithread*)**		**2 Phi + CPU (*multithread*)**	
Size	**Phi (%)**	**CPU (%)**	**Phi (%)**	**CPU (%)**
$24K \times 24K$	71	29	84	16
$32K \times 32K$	70	30	82	18