

Constant-Time Algorithm for Medial Axis Transform on the Reconfigurable Mesh*

Amitava Datta
Department of Computer Science
University of Western Australia
Perth, WA 6907
Australia
datta@cs.uwa.edu.au

Abstract

The medial axis transform (MAT) of a set S is defined as the set of centers and radii of maximal disks that are contained in S . If S is a region of a binary image, the disks are defined as upright maximal squares of black pixels and the region S is the union of these maximal squares. In this paper, we present an $O(1)$ time algorithm for computing the medial axis transform of an $N \times N$ binary image on a reconfigurable mesh of size $N \times N \times N$.

1. Introduction

Shape representation is an important problem in pattern recognition. One of the most important issues related to shape representation is data reduction. The *medial axis transform* (MAT) was first introduced by Blum [1] as a shape descriptor which has very good potential for data reduction. Since its introduction, MAT has been extensively studied in computer vision and pattern recognition literature. See the recent paper by Choi *et al* [2] for detailed references. The book by Rosenfeld and Kak [9] gives an excellent introduction to MAT.

The MAT of a set S can be defined as the set of centers and radii of the maximal disks that are contained in S . In case of a binary image, each pixel in the image can take a value either 1 (*black*) or 0 (*white*). The regions of interest in such an image are the regions of 1-pixels. The disks in a binary image are upright squares (sides parallel to the axes) of 1-pixels. The MAT of a binary image is the set of maximal squares, i.e., the set of squares such that no member of this set is enclosed by a larger square in the set. The region S

can be completely reconstructed as the union of these maximal squares. Hence, the MAT is a succinct representation of a binary image. The MAT representation may still be redundant, since some squares may be contained in the union of others. However, the problem of finding the minimum number of rectangles that cover the 1-pixels in a binary image is NP-hard [3]. Hence, it is important to detect all the maximal disks or squares of 1-pixels in order to reconstruct the image correctly.

The computation of the MAT of a binary image has been studied both in the sequential and parallel domains. Jenq and Sahni [5] presented an $O(N^2)$ sequential algorithm for this problem. However, many real-time image processing tasks require the computation of MAT and hence it is important to develop fast parallel algorithms for this problem. Jenq and Sahni [5] designed an $O(\log N)$ time, N^2 processor algorithm both for the CREW PRAM and the SIMD hypercube. Later, a work-optimal CREW PRAM algorithm was reported by Kim [6]. His algorithm runs in $O(\log N)$ time using $O(N^2/\log N)$ processors. In this paper, we present an $O(1)$ time algorithm for computing the MAT of a binary image of size $N \times N$ on a reconfigurable mesh of size $N \times N \times N$. Our algorithm has the fastest possible running time for this problem.

The *reconfigurable mesh* model was first proposed by Miller *et al* [7] as an attempt to reduce the communication diameter of a conventional mesh of processors. The reconfigurable mesh has been used recently for solving a variety of problems. These include algorithms for fundamental data movements, sorting, selection, image processing, graph problems and computational geometry.

Each processor in an $N \times N$ reconfigurable mesh has four ports which can be configured under program control. Hence, it is possible to construct long buses within the mesh to transfer data. Only one processor can write to such a bus in a clock cycle, however all other processors connected

*This work was partially supported by a Research Launching Grant of the Faculty of Engineering and Mathematical Sciences, University of Western Australia.

to the bus can read from the bus. Such a broadcast takes $\Theta(1)$ time. See the paper by Jang *et al* [4] for detailed discussion on the reconfigurable mesh architecture as well as references to many algorithms developed for the reconfigurable mesh.

We use the three dimensional right-handed xyz coordinate system for representing a three-dimensional reconfigurable mesh. A processor is indexed as $PE(i, j, k)$ where i, j and k are the indices along the x, y and z directions respectively. The origin of the coordinate system is at the top-left corner of the xy -plane and the processor at the top-left corner of the xy -plane has an index $PE(1, 1, 1)$. An $N \times N \times N$ mesh has N submeshes parallel to the xy -plane each of size $N \times N$. All the processors on the i^{th} such submesh are represented as $PE(*, *, i)$, when $PE(*, *, 1)$ is the submesh on the xy -plane. Similarly, all the processors on the j^{th} (resp. j^{th}) submesh parallel to the yz (resp. xz) plane are represented by $PE(i, *, *)$ (resp. $PE(*, j, *)$).

Given a sequence a_1, a_2, \dots, a_n , the prefix sum problem involves computing all the sums $a_1, a_1 + a_2, a_1 + a_2 + a_3, \dots, a_1 + a_2 + \dots + a_n$. Olariu *et al* [8] presented a prefix sum algorithm for a binary sequence i.e., each $a_i \in \{0, 1\}$. We need a generalization of the algorithm in [8]. In our version of this problem, each $a_i \in \{-1, 0, +1\}$. Suppose, z_j denotes the j^{th} prefix sum ($1 \leq j \leq n$), i.e., $z_j = a_1 + a_2 + \dots + a_j$. We omit the proof of the following lemma in this version.

Lemma 1.1 *Given a sequence a_1, a_2, \dots, a_n with each $a_i \in \{-1, 0, +1\}$, the prefix sums z_j for this sequence can be computed on a reconfigurable mesh of size $n \times n$ in $O(1)$ time, provided none of the prefix sums z_j is negative, i.e., $z_j \geq 0, 1 \leq j \leq n$.*

The rest of this paper is organized as follows. We discuss some properties of the medial axis transform in Section 2. The algorithm for computing the MAT of a binary image is presented in Section 3.

2. Properties of MAT

We assume that the image I is given as an $N \times N$ array with a black pixel (resp. white pixel) represented by 1 (resp. 0) in this image. The origin of the coordinate system is at the top-left corner of the image and the pixel at the top-left corner is represented as $I[1, 1]$. The x (resp. y) coordinate increases downwards (resp. towards right). This is the usual matrix representation of an image. We assume a *right-handed* coordinate system to make the notations consistent with the notations of the mesh. A pixel $I[i, j]$ is in the i^{th} row and j^{th} column of the image.

In [9], the squares in a MAT are restricted to be of odd length. However, we feel that this restriction is unnecessary and we can use any set of regular shapes like squares of odd

or even lengths or even rectangles. The main requirement is that it should be possible to represent and manipulate any such shape compactly. This approach has been taken previously in [5]. So, in this paper, we assume that the shapes are squares (of odd or even lengths) and we determine the top-left corner as well as the side lengths of these squares.

For a 1-pixel $I[i, j]$, the set of pixels $I[i + m, j + n]$ such that $m \leq (N - i)$ and $n \leq (N - j)$ and $m, n = 0, 1, 2, \dots$ is called the diagonal of $I[i, j]$ or $Diag(i, j)$. If there is a set of contiguous 1-pixels in $Diag(i, j)$ starting at $I[i, j]$, this set of contiguous 1-pixels is called the *full diagonal* of $I[i, j]$ and denoted by $FD(i, j)$. $|FD(i, j)|$ denotes the number of 1-pixels in $FD(i, j)$.

For a 1-pixel $I[i, j]$, the set of pixels $I[i + m, j], 0 \leq m \leq (N - i)$ (resp. $I[i, j + m], 0 \leq m \leq (N - j)$) is called the *column* (resp. *row*) of $I[i, j]$ and denoted by $Col(i, j)$ (resp. $Row(i, j)$). If there is a contiguous set of 1-pixels in $Col(i, j)$ (resp. $Row(i, j)$) starting at $I[i, j]$, this set of contiguous 1-pixels is called the *full column* (resp. *full row*) of $I[i, j]$ and denoted by $FC(i, j)$ (resp. $FR(i, j)$). $|FC(i, j)|$ (resp. $|FR(i, j)|$) denotes the number of 1-pixels in $FC(i, j)$ (resp. $FR(i, j)$).

For a 1-pixel $I[i, j]$ the maximal *square* of 1-pixels with $I[i, j]$ as the top-left corner is denoted by $S(i, j)$. For a 1-pixel $I[i, j]$, if there is an equilateral triangle of 1-pixels above $FD(i, j)$ and with $FD(i, j)$ as one of its sides, this triangle of 1-pixels is called the *possible upper-half* of the square $S(i, j)$ or $UH(i, j)$. Similarly, if there is an equilateral triangle of 1-pixels below $FD(i, j)$ and with $FD(i, j)$ as one of its sides, this triangle of 1-pixels is called the *possible lower-half* of the square $S(i, j)$ or $LH(i, j)$. The 1-pixels on $FD(i, j)$ are included both in $UH(i, j)$ and $LH(i, j)$. These definitions are illustrated in Figure 1. The proof of the following lemma is omitted in this version.

Lemma 2.1 *For $k > j$, a 1-pixel $I[i, k] \in UH(i, j)$ iff for every pixel $I[i, m], j \leq m \leq k, |FC(i, m)| \geq (m - j + 1)$. Similarly, for $k > i$, a 1-pixel $I[k, j] \in LH(i, j)$ iff for every pixel $I[m, j], i \leq m \leq k, |FR(m, j)| \geq (m - i + 1)$.*

If $I[i, n], n > j$ is the last 1-pixel in the i^{th} row such that $I[i, n] \in UH(i, j)$, we say that $UH(i, j)$ has a *side-length* $n - j + 1$ and it is represented as $|UH(i, j)|$. Similar to $|UH(i, j)|$, we denote the side-length of $LH(i, j)$ as $|LH(i, j)|$.

Lemma 2.2 *The maximal square of 1-pixels with top-left corner as $I[i, j]$ has a side-length $\min(|UH(i, j)|, |LH(i, j)|)$.*

3. Algorithm for computing the MAT

There are two phases in our algorithm. In the first phase, for every 1-pixel $I[i, j], 1 \leq i \leq N, 1 \leq j \leq N$, we

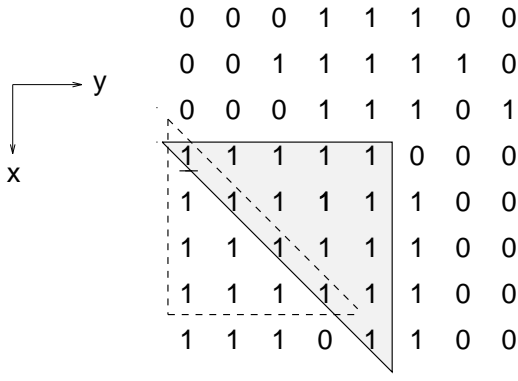


Figure 1. An 8×8 binary image and illustration for some of the definitions. For the underlined 1-pixel $I[4, 1]$, $|FD(4, 1)| = 5$, $|FC(4, 1)| = 5$ and $|FR(4, 1)| = 5$. The shaded triangle is $UH(4, 1)$ and the dashed triangle is $LH(4, 1)$. $|UH(4, 1)| = 5$ and $|LH(4, 1)| = 4$. Hence, $|S(4, 1)| = \min(|UH(4, 1)|, |LH(4, 1)|) = 4$.

find the maximal square of 1-pixels with $I[i, j]$ as the top-left corner. There may be many squares of 1-pixels with $I[i, j]$ as the top-left corner, however, our aim in the first phase is to find the largest square among those. We assume that initially the image is given in the processors $P(*, *, 1)$, one pixel per processor. In other words, the pixels $I[i, j]$, $1 \leq i \leq N, 1 \leq j \leq N$ are given in the processor $PE(i, j, 1)$, $1 \leq i \leq N, 1 \leq j \leq N$. If we omit the third index of the mesh, it is assumed that the computation is done in the two dimensional mesh on the xy -plane, i.e., in the mesh $PE(*, *, 1)$.

3.1. Phase 1

Step 1.

In this step, each processor $P(i, j)$ holding a 1-pixel, computes $|FC(i, j)|$. Each processor that holds a 1-pixel, connects its N and S ports. If a processor holds a 0-pixel, it does not connect these two ports. The mesh $P(*, *)$ is now split into disjoint buses parallel to the x -axis. If a processor $P(m, n)$ is the last processor on such a bus, it writes its index on the bus that starts at processor $P(m - 1, n)$. A processor $PE(i, j)$ on this bus reads the index of the last processor on the bus and can determine $|FC(i, j)|$ from this index. This computation is done simultaneously for all such buses and takes $O(1)$ time.

Step 2.

The next task is to determine $|FD(i, j)|$ for all the processors $PE(i, j)$ that hold a 1-pixel. This is done in the following way. Each processor $PE(i, j)$ that holds a 1-

pixel, connects its two pairs of ports, N,E and W,S. If a processor holds a 0-pixel, it connects only the W,S pair of ports. This can be done in $O(1)$ time for all the processors. Now the mesh $PE(*, *)$ is divided into disjoint diagonal buses. Each processor $PE(m, n)$ holding a 1-pixel now determines whether it is the last processor on such a bus. This is done by checking whether $PE(m + 1, n + 1)$ holds a 1 or a 0-pixel. $PE(m, n)$ is the last processor on a bus if $PE(m + 1, n + 1)$ holds a 0 pixel. If $PE(m, n)$ is a last processor on its bus, it writes its index on the bus that starts at the N port of $PE(m, n)$. A processor $PE(i, j)$ on the bus reads this index from the bus that connects the N and E ports of $PE(i, j)$ and can determine $|FD(i, j)|$. This computation is done simultaneously for all the buses and takes $O(1)$ time.

The computation of $|FR(i, j)|$ is similar to Step 1 and can be done through construction of disjoint horizontal buses in $O(1)$ time. At the end of Step 2, each processor $PE(i, j)$ holding a 1-pixel gets the three values $|FD(i, j)|$, $|FC(i, j)|$ and $|FR(i, j)|$. Note that, $|FD(i, j)|$ and $|FC(i, j)|$ are required for computing $UH(i, j)$ and $|FD(i, j)|$ and $|FR(i, j)|$ are required for computing $LH(i, j)$. In the following step, we discuss the method of computing $UH(i, j)$. The computation of $LH(i, j)$ is similar.

Step 3.

We compute $UH(i, j)$ from $|FD(i, j)|$ and $|FC(i, j)|$ in this step. For each pixel $PE(i, j)$ holding a 1-pixel, we have to check the condition in Lemma 2.1 for all the pixels $PE(i, k)$, $j < k \leq N$. We describe the computation for the i^{th} row of the image, but the same computation is done simultaneously in all the rows.

For the i^{th} row $PE(i, *, 1)$, we use the i^{th} mesh $PE(i, *, *)$ parallel to the yz -plane. A processor $PE(i, j, 1)$ sends $|FD(i, j)|$ and $|FC(i, j)|$ to the processors $PE(i, j, m)$, $N - j + 1 \leq m \leq N$. Note that, only the upper triangular processors (including the main diagonal) of the mesh $PE(i, *, *)$ get data elements due to this transmission. All the processors that receive some input due to this transmission are called *active* and other processors are called *inactive*. Note that, among all the rows of $PE(i, *, *)$ to which $|FC(i, j)|$ and $|FD(i, j)|$ have been transmitted, the row $PE(i, *, j)$ holds these two quantities in the first active processor $PE(i, j, j)$ in that row. The computation in $PE(i, *, j)$ is as follows. $PE(i, j, j)$ broadcasts $|FD(i, j)|$ to all the processors to its right i.e., to the processors $PE(i, n, j)$, $j < n \leq N$. This is done through the construction of a bus by joining the E and W ports of the processors in this row. Consider a processor $PE(i, k, j)$, $k > j$. Note that, $PE(i, k, j)$ has already received $|FC(i, k)|$ in the previous step. The 1-pixel originally in $PE(i, k)$ is part of $UH(i, j)$ if $|FC(i, k)|$ satisfies the condition in Lemma 2.1. $PE(i, k, j)$ can verify this in

$O(1)$ time from $|FD(i, j)|$ and $|FC(i, k)|$. If $PE(i, k, j)$ is part of $UH(i, j)$, it stores 1 in one of its registers R_1 . Otherwise, it stores 0 in R_1 .

$|UH(i, j)|$ is determined in the following way. A processor that holds a 1 in its R_1 register, connects its E and W ports. If a processor holds 0 in its R_1 register, it does not connect these two ports. Now, the processors $PE(i, n, j), j < n \leq N$ are divided into disjoint buses. If a processor $PE(i, k, j)$ is the last processor in its bus, it writes its index on the bus towards its left, i.e., the bus starting at $PE(i, k-1, j)$. The index received by $PE(i, j, j)$ is the index of the last processor $PE(i, m, j), m > j$ which is part of $UH(i, j)$. $PE(i, j, j)$ can compute the side length of $UH(i, j)$ from this index. This computation takes $O(1)$ time and is done simultaneously in parallel for all the rows of the mesh $PE(i, *, *)$. At the end of this step, the quantities $|UH(i, j)|, 1 \leq i, j \leq N$ are sent back to the processors $PE(i, j, 1)$ through disjoint buses perpendicular to the xy -plane. The computation of $LH(i, j)$ is similar and is done by using the meshes parallel to the xz plane.

At the end of Step 3, each processor $PE(i, j)$ has got both the quantities $|UH(i, j)|$ and $|LH(i, j)|$ and can determine $S(i, j)$ using the condition of Lemma 2.2.

3.2. Phase 2

Some of the squares found in Phase 1 may not be maximal in the global sense. This is due to the fact that if a square $S(i, j)$ encloses a square $S(k, l), i \leq k, j \leq l$ and $S(i, j) \neq S(k, l), S(k, l)$ should not be reported as part of the MAT for the image. Our task in this phase is to eliminate all the squares that are not globally maximal, i.e., squares which are enclosed by larger squares.

We do this elimination in two subphases. In the first subphase, Phase 2.1, we eliminate a non-maximal square $S(i, k)$ if the square that encloses $S(i, k)$ has its top-left corner in the i^{th} row or in the k^{th} column.

A square S_j may be enclosed completely by a maximal square S_i , or by a collection of maximal squares $S_m, S_{m+1}, \dots, S_{m+n}$. In this case, the top-left corners of the squares that collectively enclose S_i are in different rows and columns (with respect to the top-left corner of S_j). We eliminate all such squares S_j in the second subphase, Phase 2.2.

3.2.1. Phase 2.1

Step 4.

We determine the first type of enclosure in this step. For the i^{th} row, we use the mesh perpendicular to the i^{th} row and parallel to the yz -plane. In our terminology, the enclosure of squares with their top-left corners in processors $PE(i, j, 1), 1 \leq j \leq N$ are determined in the mesh $PE(i, *, *)$ which is parallel to the yz plane.

Each processor $PE(i, j, 1), 1 \leq j \leq N$ sends $|S(i, j)|$ to the processors $PE(i, j, n), 1 \leq n \leq N - j + 1$. Since $S(i, j)$ can be enclosed by some other square $S(i, k), k < j$, the top-left corner of $S(i, k)$ is to the left of that of $S(i, j)$. So, it is sufficient to consider only the squares $S(i, k), 1 \leq k < j$ to see whether at least one of those squares encloses $S(i, j)$. Note that, the j^{th} row of the mesh $PE(i, *, *)$, i.e., $PE(i, *, j)$ has $S(i, n), 1 \leq n \leq j$ in the processors $PE(i, n, j), 1 \leq n \leq j$ due to the data transmission above. Hence, we use the processors in $PE(i, *, j)$ to determine whether $S(i, j)$ is enclosed by any other square $S(i, k), k < j$. This is done in the following way. $PE(i, j, j)$ (holding $S(i, j)$) broadcasts $S(i, j)$ to the processors $PE(i, n, j), 1 \leq n < j$. A processor $PE(i, m, j), m < j$ holds $S(i, m), m < j$ and can determine in $O(1)$ time whether $S(i, m)$ completely encloses $S(i, j)$. If $S(i, m)$ completely encloses $S(i, j)$, $PE(i, m, j)$ writes 1 in one of its registers R_2 , otherwise, it writes 0 in R_2 . After this, a processor $PE(i, n, j), n < j$ connects its E and W ports if its R_2 register holds 0. If $PE(i, n, j)$ holds 1 in its R_2 register, it does not connect its E and W ports. Now, the row $PE(i, *, j)$ is split into a set of disjoint buses if at least one $S(i, m), m < j$ completely encloses $S(i, j)$. The last processor of each such bus writes its index onto the bus towards its right. If $PE(i, j, j)$ gets an index from its left, it is completely enclosed by at least one square $S(i, m), m < j$.

This computation is done in each row of the mesh $PE(i, *, *)$ and in all the meshes $PE(i, *, *), 1 \leq i \leq N$. Hence, after this step, each square $S(i, j), 1 \leq i, j \leq N$ that is enclosed by another square $S(i, m), m < j$ is eliminated. The elimination of squares of the second type is similar to Step 4 and the details are omitted in this version.

3.2.2. Phase 2.2

The task in this phase is to eliminate the remaining non-maximal squares. After Phase 1, a processor $PE(i, j, 1)$ that has the top-left pixel of a maximal square knows the square $S(i, j)$ along with its side length $|S(i, j)|$. However, we need the processor at the bottom-left corner of $S(i, j)$ also to know $|S(i, j)|$ for the computation in this phase. This information can be passed to the relevant processors through construction of disjoint buses in $O(1)$ time and the details are omitted.

We also need to mark all the processors in $PE(*, *, 1)$ to indicate whether each of them is part of a top or a bottom side of a square. In our computation, we need to mark a processor $PE(i, j, 1)$ with +1 if it is part of at least one top side, with -1 if it is part of at least one bottom side and with 0 if it not a part of any top or bottom side. This computation can be done in $O(1)$ time for all the processors $PE(i, j, 1)$ and the details are omitted in this version.

Step 5.

The purpose of this step is to perform a parallel prefix computation to determine whether a 1-pixel is part of only one square or part of multiple squares. We discuss this computation for the j^{th} column, i.e., for processors $PE(*, j, 1)$. This computation is done simultaneously in all the columns. The prefix sum is computed from $PE(1, j, 1)$ down to $PE(N, j, 1)$. The mesh used for this computation is $PE(*, j, *)$, i.e., the j^{th} mesh parallel to the xz -plane. The algorithm in Lemma 1.1 is used for this computation and at the end each processor $PE(*, j, 1)$ gets their prefix values in their R_1 registers. Note that, we can use Lemma 1.1 since none of the prefix sums can be negative. This is due to the fact that each bottom side must have a corresponding top side that appears earlier. So, for every prefix in the sequence of +1, 0 and -1, the number of +1s is at least equal to the number of -1s. After this computation, if a processor $PE(i, j, 1)$ holds some integer $> +1$ in its R_1 register, the 1-pixel in it is enclosed by more than one square and if it holds +1, the 1-pixel in it is enclosed by or part of only one square. The time requirement for this step is $O(1)$ since the computation in Lemma 1.1 takes $O(1)$ time.

Step 6.

The task in this step is to finally eliminate all the remaining non-maximal squares. This computation is done in the mesh $PE(i, j, 1)$, i.e., the two dimensional mesh on the xy -plane. Each processor $PE(k, m, 1)$ (holding a 1-pixel) that holds a value +1 in its R_1 register after Step 12, connects its E and W ports. All other processors (either holding $> +1$ in R_1 register or holding a 0 pixel) do not connect their E and W ports. Now a processor $PE(k, m, 1)$ that holds +1 in its R_1 register, writes its index on the bus starting at $PE(k, m - 1, 1)$. If a processor $PE(k, j, 1)$, $j < m$, gets such an index \mathcal{I} , it stores this index in a register R_3 and marks itself with a special symbol \$.

Consider a square $S(i, j)$ with side length $|S(i, j)|$. If at least one of the processors $PE(m, j, 1)$, $i < m \leq |S(i, j)|$ has got an index \mathcal{I} of a processor that is within $S(i, j)$, due to the data transmission above, that means $S(i, j)$ has at least one pixel that is not enclosed by any of the other squares and hence $S(i, j)$ is not completely enclosed by any other square. If none of the processors $PE(m, j, 1)$, $i < m \leq |S(i, j)|$ gets such an index \mathcal{I} due to the data movement above, it means $S(i, j)$ is completely enclosed by at least one larger maximal square (or by a collection of maximal squares).

The remaining part of this computation is done in the following way. Each processor in the j^{th} column, i.e., processors $PE(m, j, 1)$, $1 \leq m \leq N$ connects its N and S ports if it is not marked by \$ and it had a 1-pixel of the image initially. If it is marked by \$, it does not connect its N and S ports. Also, it does not connect N and S ports if it had originally a 0-pixel of the image. Now, a processor

$PE(n, j, 1)$ that is marked by \$ and has got an index \mathcal{I} , writes \mathcal{I} on the bus (parallel to the x -axis) that starts at the processor $PE(n - 1, j, 1)$. A processor $PE(i, j, 1)$ (which is the top-left corner of the square $S(i, j)$) reads this index. $PE(i, j, 1)$ can decide whether the index \mathcal{I} is within $S(i, j)$. If so, there is at least one pixel in $S(i, j)$ which is not enclosed by any larger square and hence $S(i, j)$ is not completely enclosed by any larger square. If either \mathcal{I} is not within $S(i, j)$ or $PE(i, j, 1)$ does not receive any index from a processor like $PE(n, j, 1)$, all the pixels in $S(i, j)$ are enclosed by at least one larger square. Note that, a non-maximal square can be collectively enclosed by a set of maximal squares. This computation is done simultaneously in all the processors $PE(i, j, 1)$, $1 \leq i \leq N$ that hold the top-left corner pixels of the squares and also in each of the columns $PE(i, j, 1)$, $1 \leq j \leq N$ of the mesh $PE(*, *, 1)$. This computation takes $O(1)$ time.

After this step, only the processors that hold the top-left corners of maximal squares (i.e., not completely enclosed by other maximal squares) mark themselves as maximal squares.

Theorem 3.1 *The medial axis transform of an $N \times N$ binary image can be computed in $O(1)$ time on a reconfigurable mesh of size $N \times N \times N$.*

References

- [1] Blum, H. *Models for the Perception of Speech and Visual Form*. Cambridge, MA: MIT Press, 1967, pp. 362-380.
- [2] Choi, H. , Choi, S. , Moon, H. and Wee, N. New algorithms for medial axis transform of plain domain. *Graphical Models and Image Processing*, **59**, 6, (1997), 463-483.
- [3] Garey, M. J. and Johnson, D. S. *Computers and Intractability*. San Francisco: W. H. Freeman, 1979.
- [4] Jang, J. , Nigam, M. , Prasanna, V. K. and Sahni, S. Constant time algorithms for computational geometry on the reconfigurable mesh. *IEEE Trans. Parallel and Distributed Systems*, **8**, (1997), 1-12.
- [5] Jenq, J. and Sahni, S. Serial and parallel algorithms for the medial axis transform. *IEEE Trans. Pattern Analysis and Machine Intelligence*, **14**, 12, (1992), 1218-1224.
- [6] Kim, S. K. Optimal parallel algorithms for region labeling and medial axis transform of binary images. *SIAM Journal on Discrete Mathematics*, **4**, 3, (1991), 385-396.
- [7] Miller, R. , Prasanna Kumar, V. K. , Reisis, D. and Stout, Q. F. Parallel computations on reconfigurable meshes. *IEEE Trans. Computers*, **42**, (1993), 678-692.
- [8] Olariu, S. , Schwing, J. , and Zhang, J. Data movement techniques on reconfigurable meshes, with applications. *International J. High Speed Computing*, **6**, 2, (1994), 311-323.
- [9] Rosenfeld, A. and Kak, A. *Digital Picture Processing*, Academic Press, 1982.