# Determining Maximum $k$-Width-Connectivity on Meshes [*]

Susanne E. Hambrusch [†]
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA

Frank Dehne [‡]
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6

February 16, 1996

**Abstract**

Let $I$ be a $n \times n$ binary image stored in a $n \times n$ mesh of processors with one pixel per processor. Image $I$ is *k-width-connected* if, informally, between any pair of 1-pixels there exists a path of width $k$ (composed of 1-pixels only). We consider the problem of determining the largest integer $k$ such that $I$ is k-width-connected, and present an optimal $O(n)$ time algorithm for the mesh architecture.

**Keywords**: Parallel algorithms, meshes, binary images, connected components, k-width-connectivity.

[*]A preliminary version of the paper appeared in the *Proceedings of 6-th International Parallel Processing Symposium,* Beverly Hills, CA, March 1992.

[†]Research supported in part by ONR under contracts N00014-84-K-0502 and N00014-86-K-0689, and by NSF under Grant MIP-87-15652.

[‡]Research partially supported by the Natural Sciences and Engineering Research Council of Canada.

## 1 Introduction

Detecting forms of connectivity in binary images is a fundamental problem in image processing [12, 14]. Because of the relevance of parallel computation to image processing and computer vision, the parallel complexity of connectivity problems has been studied extensively [2, 3, 5, 6, 7, 8, 11]. In this paper we consider k-width-connectivity which is a stronger, more fault-tolerant form of connectivity in binary images. Informally, an image is k-width-connected if between any pair of 1-pixels (i.e., entries of value '1') there exists a path of width $k$. In [4], Dehne and Hambrusch present a parallel mesh algorithm that, given an integer $k$ and a binary image $I$ stored in an $n \times n$ mesh with one pixel per processor, determines the k-width-components of $I$ in $O(n)$ time. The problem of determining the k-width-components has a number of applications. One is in image segmentation where an image is partitioned into coherent regions that satisfy certain requirements [12]. Another application is the detection of connectivity in VLSI masks where electrical connectivity between components is maintained by a channel whose width is never less than a value $\lambda$ [9]. The image might also represent the corridors of a maze, in which case the fact that $a$ and $b$ are in the same k-width-component implies that a robot occupying a $k \times k$ area is able to move from $a$ to $b$.

A natural problem that arises is that of, given a binary image $I$, determine the largest integer $k$ so that image $I$ is k-width-connected (i.e., $I$ contains one k-width-component). For the remainder of the paper let $k^*$ denote this largest $k$. Determining $k^*$ has obvious relevance to the applications stated above. For example, if the image represents the corridors of a maze, then $k^*$ represents the largest side length of a square-shaped robot that can move freely between any two points in the maze.

The value of $k^*$ can be determined in $O(n \log k^*)$ time by using the algorithm presented in [4] and performing a binary search for $k^*$. In this paper we present an algorithm to determine $k^*$ in $O(n)$ time. Our algorithm is based on a very different approach from the one used in [4]. We generate $k^*$ in two stages. The first stage generates a preliminary estimate for $k^*$ by having every 1-pixel perform "local" computations. This preliminary value represents an upper bound on the value of $k^*$. The final value of $k^*$ is obtained by generating a weighted graph that models bottlenecks in image $I$. For this graph we determine the largest edge weight $k'$ such that removing all edges of weight at least $k'$ breaks all cycles in the graph. We then show that $k'$ equals $k^*$.

The paper is organized as follows. In Section 2 we state some of the necessary definitions. Section 3 contains the description of the approach used by our algorithm and Section 4 describes its implementation on the mesh. Section 5 concludes.

## 2 Definitions and Preliminaries

Throughout, image $I$ is of size $n \times n$ and is stored in a mesh containing $n^2$ processors, with every processor containing $O(1)$ registers. We assume that the image is stored in the obvious way; i.e., the processor in row $i$ and column $j$ stores the pixel in the same row and column. In cases where it is obvious, we refer to the processor storing pixel $x$ as processor $x$.

We start by giving the formal definition of a k-width-connected image.

We assume, w.l.o.g., that image $I$ contains no 1-pixels in the first and last rows and columns. Two pixels are called *hv-adjacent* if they are horizontally or vertically adjacent and they are called *d-adjacent* if they are diagonally adjacent. A *1-block* (of size $k$) is a subimage of $I$ of size $k \times k$ which contains only 1-pixels. Let $x$ and $y$ be two 1-pixels in image $I$. There exists a path $P(x, y)$ from $x$ to $y$ if and only if there exist 1-pixels $x = v_0, v_1, \ldots, v_{m-1}, v_m = y$ such that $v_i$ and $v_{i+1}$ are hv-adjacent, $v_i \neq v_j$ for $i \neq j$.

A *path of width* $k$ is a sequence $B_1, \ldots, B_l$ of 1-blocks, $l \geq 1$, each of size $k$, such that $B_i$ and $B_{i+1}$ share a subimage of size $(k-1) \times k$ or $k \times (k-1)$, $1 \leq i \leq l-1$. A path of width 3 is shown in Figure 1.

Two 1-pixels $a$ and $b$ are *k-width-connected* if and only if there exists a path of width $k$ containing both $a$ and $b$. Image $I$ is *k-width-connected* if and only if any two 1-pixels $a$ and $b$ of $I$ are k-width-connected. Figure 2 shows an image that is 3-width-connected, but not 4-width-connected. The image consists of two 4-width-components: one is formed by the 1-pixels "enclosed" by holes $H_1$ and $H_2$ and one is formed by the remaining 1-pixels with seven 1-pixels belonging to both components. Observe that a 1-pixel can belong up to $k$ k-width-components.

To reduce the number of special cases that need to be considered when determining $k^*$, we assume $I$ to be 2-width-connected. Using the algorithm presented in [4], we can determine in $O(n)$ time whether $I$ is 2-width-connected.

We characterize 1-pixels that are hv- or d-adjacent to at least one 0-pixel into contour and corner pixels. A 1-pixel hv-adjacent to exactly one 0-pixel is a *1-contour* pixel and a 1-pixel hv-adjacent to exactly two 0-pixels is a *2-*

*contour* pixel. Since $I$ is at 2-width-connected, a 1-pixel can be hv-adjacent to at most two 0-pixels and these two 0-pixels cannot be in the same row or column. A 1-pixel $x$ is a *1-corner* pixel if $x$ is d-adjacent to exactly one 0-pixel $p$ with the following corner property: $x$ and $p$ are hv-adjacent to two common 1-pixels (both of which are contour pixels). For any 1-pixel $x$ there can be two distinct 0-pixels making $x$ a corner pixel. If $x$ is d-adjacent to two such 0-pixels, we say $x$ is a *2-corner* pixel. Note that the two 0-pixels making $x$ a 2-corner pixel cannot be in the same row or column. A 1-pixel can be both a 1-contour pixel and a 1-corner pixel. However, since $I$ is 2-width-connected, it cannot be both a 2-contour and a 1-corner pixel (the combination 1-contour and 2-corner is also not possible). See Figure 2 for an illustration of the different types of 1-pixels.

The *boundary graph* $G_b = (V_b, E_b)$ is an undirected, planar graph with weights on both the vertices and the edges. For any vertex $x$, let $w(x)$ be the weight of vertex $x$ and for every edge $(x, y)$, let $w(x, y)$ be its weight. Every contour pixel $x$ corresponds to one vertex in $G_b$, namely contour vertex $x$, with $w(x) = \infty$. Let $x$ and $y$ be two hv-adjacent contour pixels such the two 0-pixels making $x$ and $y$ contour pixels are also hv-adjacent. Then, $G_b$ contains the edge $(x, y)$ whose weight equals the size of the largest 1-block that contains both $x$ and $y$. This 1-block has $x$ and $y$ on its border. A contour vertex is incident to exactly two edges. For a contour vertex hv-adjacent to a corner vertex (and thus hv-adjacent to only one contour vertex), the second edge is formed by the rules described below.

Every 1-corner pixel $x$ induces one vertex $x$ in $G_b$. Let $y$ and $z$ be the two 1-pixels hv-adjacent to pixel $x$ and which are also hv-adjacent to the 0-pixel making $x$ a corner pixel. The weight of vertex $x$, $w(x)$, equals the size of the largest 1-block that contains pixel $x$ in one of its four corners, but does *not* contain $y$ nor $z$. Graph $G_b$ contains the edges $(y, x)$ and $(x, z)$. The weight of the edge $(y, x)$ (resp. $(x, z)$) equals the size of the largest 1-block containing both $y$ and $x$ (resp. $x$ and $z$). Intuitively, $\min\{w(x), w(y, x), w(x, z)\}$ is the maximum width of a path from 1-pixel $y$ to 1-pixel $z$ via 1-pixel $x$. Clearly, the image may contain another, wider, path from $y$ to $z$. When a pixel $x$ is both a 1-contour and a 1-corner pixel it induces two distinct vertices in $G_b$. When pixel $x$ is a 2-corner pixel, $x$ also induces two vertices in $G_b$, one for each corner. The weights and the adjacent edges are set in a corresponding way.

Figure 3 shows the boundary graph induced by the image of Figure 2. In Figure 3 corner vertices are represented by solid circles and contour vertices by empty circles. The weights are shown only for edges and vertices in the connected components induced by holes $H_4$ and $H_5$ (weights of $\infty$ are not shown). Three pairs of vertices are enclosed by dashed lines; each such pair is induced by a single 1-pixel.

Let $B_1$ be the set of 1-pixels corresponding to the exterior boundary of the component in image $I$ and let $B_2, B_3, \ldots, B_m$ be the $m - 1$ interior boundaries, $m \geq 1$. The boundary graph $G_b$ consists of $m$ connected components, each having the form of a cycle. The two vertices induced by a pixel that is a 1-contour and a 1-corner pixel (resp. a 2-corner pixel) belong to different connected components (since $I$ is 2-width-connected, such 1-pixels are part of two different boundaries).

## 3 Overview of the Algorithm

As stated in the introduction, the value of $k^*$ is determined in two stages. In the first stage we determine the boundary graph $G_b$ and compute an upper bound on $k^*$ by using the weights associated with the edges of the boundary graph. Let $k$ be the value generated by the first stage, $k \geq k^*$. In the second stage we use the value of $k$ to set up a hole graph that models the bottlenecks of size at most $k-1$ induced by corner pixels. We determine $k^*$ by applying a cycle-breaking procedure to the hole graph. Our algorithm makes use of the following two properties.

**Property 1** *Let $x$ and $y$ be two adjacent contour vertices such that $G_b$ contains the edge $(x, y)$. Then, $k^* \leq w(x, y)$.*

**Proof:** By definition of the edges of the boundary graph, the largest 1-block containing both pixel $x$ and pixel $y$ has size $w(x, y)$. Assume image $I$ is $k^*$-width-connected with $k^* > w(x, y)$. Then, there must exist two disjoint 1-blocks of size $k^*$ such that one 1-block has $x$ in one of its corners and the other 1-block has $y$ in one of its corners. However, this implies that there exists a 1-block of size $k^*$ containing both $x$ and $y$. This is not possible and thus the property follows. $\square$

For any vertex in $G_b$ representing a contour or corner pixel $x$, we denote the two vertices in $G_b$ adjacent to $x$ (as well as the respective pixels) by $a1(x)$ and $a2(x)$.

**Property 2** *Let $x$ be a contour vertex adjacent to a corner vertex in $G_b$. Then, $k^* \leq \max\{w(x, a1(x)), w(x, a2(x))\}$.*

**Proof:** Let $x$ be such a contour vertex. If image $I$ is $k^*$-width-connected, then 1-pixel $x$ is contained in at least one 1-block of size $k^*$. Since $k^* \geq 2$, at least one of $a1(x)$ and $a2(x)$ is in this 1-block of size $k^*$. $\square$

**STAGE 1**

The first stage of our algorithm generates a value $k \geq k^*$ for which Properties 1 and 2 are satisfied. For the boundary graph of Figure 3, Stage 1 determines $k = 3$ (which coincides with the final answer). Using $k$, the contour pixels of image $I$ can be partitioned into sets so that between any two contour pixels in the same set there exists a path of width $k$. Assume we have generated such a partition into the minimum number of sets. If this partition consists of only one set, we have $k = k^*$. Otherwise, some corner pixels induce bottlenecks that force $k^* < k$. Let $x$ be a corner vertex with $y = a1(x)$ and $z = a2(x)$. The path from $y$ to $z$ via $x$ in image $I$ has a width of $\min\{w(x), w(x,y), w(x,z)\}$. We next prove a lemma which implies that, after after having computed $k$ to satisfy Properties 1 and 2, only the value of $w(x)$ determines the width of the path going from $y$ to $z$ via $x$.

**Lemma 1** *Let $x$ be a corner vertex and let $y$ be a contour vertex adjacent to $x$ in $G_b$. If $w(x) > w(x,y)$, then $k \leq w(x,y) < w(x)$, where $k$ is the quantity determined in Stage 1.*

**Proof:** Let $t = a1(y)$ with $t \neq x$. Since $y$ is a contour vertex in $G_b$, by Property 2 we have $k \leq \max\{w(y,x), w(y,t)\}$. If $w(y,x) \geq w(y,t)$, then $k \leq w(y,x) < w(x)$. We next show that $w(y,x) < w(y,t)$ is not possible. W.l.o.g let the pixels corresponding to vertices $x$, $y$, and $t$ be in a common row and let $x$ be to the left of $y$. Since $w(x) > w(x,y)$, there exists a 0-

pixel $p$ in the column containing pixel $y$ that is hv-adjacent to a pixel in the 1-block of size $w(x)$. (If such a 0-pixel $p$ would not exist, we would have $w(x,y) \geq w(x)$.) See Figure 4 for an illustration. Pixel $p$ limits the size of a 1-block containing $y$ and $t$ and makes it impossible to have $w(y,t) > w(y,x)$. The lemma follows. □

## STAGE 2

Let $k$ be the quantity determined in Stage 1. ¿From Lemma 1 it follows that, if $w(x) < k$, then the widest path from $a1(x)$ to $a2(x)$ going via corner pixel $x$ has a width of $w(x)$. We say that $x$ *induces a bottleneck* of size $w(x)$. If the only way to go from $a1(x)$ to $a2(x)$ is via corner pixel $x$, then this bottleneck cannot be avoided and we have $k^* \leq w(x) < k$. In order to determine which bottlenecks can and which cannot be avoided, we perform a cycle-breaking procedure on a hole graph induced by image $I$ and $k$. We next define this hole graph.

Every boundary in image $I$ induces a *hole*, with the outer boundary inducing the *outer hole* and every other boundary inducing an *inner hole*. Assume we have labeled the holes so that $H(y)$ is the label of the hole containing 0-pixel $y$. Image $I$ contains $m$ boundaries and thus $m$ holes. The *hole graph* $G_h = (V_h, E_h)$ is an undirected, planar, non-simple (i.e., it can contain multiple edges and self-loops), $m$-vertex graph with costs on the edges. Vertex $v_i$ of the hole graph corresponds to the $i$-th hole. We next describe how the edges of $G_h$ are formed. Let $x$ be a corner vertex with $w(x) < k$. The 1-block of size $w(x)$ that has pixel $x$ in one of its four corners is also called the 1-block *associated* with pixel $x$.

Let $s$ be the 0-pixel d-adjacent to corner pixel $x$. Let $t$ be a 0-pixel hv- or d-adjacent to the border of the 1-block associated with $x$ and which limits the size of this 1-block to $w(x)$. See Figure 5(a) for an illustration. Pixel $t$ cannot be in the row or column containing 0-pixel $s$. Furthermore, the following holds. There cannot exist two 0-pixels $t_1$ and $t_2$ such that $t_1$ and $t_2$ are hv-adjacent to different borders of the 1-block associated with $x$. If this would happen, there would exist a contour pixel forcing $k \leq w(x)$ in Stage 1. At this point pixel $t$ may not yet be uniquely defined. Consider the sequence $\lambda$ formed by the $w(x) + 1$ pixels adjacent to the side of the 1-block associated with $x$ and containing pixel $t$, listing as the first element the pixel in either the same row or column with pixel $x$. For the pixel labeled $x$ in Figure 5(a), we have $\lambda = 10000$. Should $\lambda$ contain more than one 0-pixel, we choose $t$ to correspond to the first 0-pixel in sequence. Observe that $\lambda$ is of the form $\{1\}^*\{0\}^+\{1\}^*$; i.e., it contains at least one 0-pixel (by definition) and it cannot contain a 1-pixel that is to the left as well to the right of a 0-pixel in $\lambda$. For example, for $w(x) = 4$, $\lambda = 01101$ is impossible. 0-pixel $t$ is always d-adjacent to a corner pixel $y$ so that $y$ belongs either to the 1-block associated with $x$, or $y$ is in the same row or column as 0-pixel $s$. It is possible that there exist two choices for pixel $y$. In this case, select $y$ so that it is closer to the row or column containing pixel $x$. Observe that we have $w(x) = w(y)$. ($w(y) > w(x)$ is not possible and $w(y) < w(x)$ would imply the existence of a contour pixel forcing $k \leq w(x)$ in Stage 1.) When corner pixel $y$ applies the rules described above to itself, $y$ chooses pixels $s$ and $x$ (as its $t$ and $y$).

Having chosen $x, s, t$, and $y$ according to these rules, we add to $G_h$ the edge $(H(s), H(t))$ with cost $w(x) = w(y)$. We also say that corner pixels $x$ and $y$ induce the edge $(H(s), H(t))$ with cost $w(x)$. Figure 5(b) shows the edges for the portion of the image shown in Figure 5(a). In summary, the hole graph is not necessarily connected, it can contain multiple edges (even with identical costs) and self-loops, and the cost of every edge is less than $k$. The next lemma characterizes how the edges of $G_h$ determine $k^*$.

**Lemma 2** *If the hole graph $G_h$ contains no cycles, then $k^* = k$. If the hole graph $G_h$ contains cycles, let $k'$ be the largest integer such that when all edges of cost $\geq k'$ are removed from $G_h$, the resulting graph contains no cycle. Then, $k^* = k'$.*

**Proof:** We first show that when $G_h$ contains no cycles (i.e., $G_h$ is a forest), there exists a path of width $k$ between any two 1-pixels. We start by proving that between any two 1-pixels d- or hv-adjacent to 0-pixels belonging to the same hole there exists a path of width $k$. The following observation is crucial. Let $x$ be a corner pixel with $w(x) < k$. Any contour pixel is contained in a 1-block of size $k$, and thus $a1(x)$ (resp. $a2(x)$) is in a 1-block of size $k$. In addition, there exists a 1-block of size $k$ containing either $x$ and $a1(x)$ or $x$ and $a2(x)$. This holds, since if neither $x$ and $a1(x)$ nor $x$ and $a2(x)$ were in a 1-block of size $k$, there would exist a contour pixel not contained in a 1-block of size $k$. Throughout the proof we assume that $x$ and $a1(x)$ are in a common 1-block of size $k$.

Assume the edges of every tree of $G_h$ are rooted towards a root, where the root is chosen arbitrarily. We prove the claim by an inductive argument. Let

$v_i$ be a leaf in a rooted tree and assume $v_i$ represents hole $H_i$. Let $< v_i, v_j >$ be the arc incident to vertex $v_i$ in the rooted tree, with $v_j$ representing hole $H_j$. Let $x_i$ and $x_j$ be the two corner pixels that induce the edge $(v_i, v_j)$ in the hole graph. Since $v_i$ is a leaf node, $x_i$ is the only corner pixel d-adjacent to a 0-pixel in $H_i$ having $w(x_i) < k$. A path of width $k$ from $x_i$ to $a2(x_i)$ containing all the 1-pixels d- or hv-adjacent to hole $H_i$ can be constructed as follows. Every contour pixel (resp. corner pixel different from $x_i$) that is hv-adjacent (resp. d-adjacent) to a 0-pixel in $H_i$ is in a 1-block of size $k$. These 1-blocks, together with the 1-block containing $a1(x_i)$ and $x_i$ form a path of width $k$ from $x_i$ to $a2(x_i)$. The claim now follows for hole $H_i$.

Let $v_i$ be a non-leaf vertex in a rooted tree and assume the claim holds for every hole corresponding to a vertex in the subtree rooted at $v_i$, excluding $v_i$. Let $< v_k, v_i >$ be an incoming arc for vertex $v_i$ and let $x_k$ and $x_i$ be the two corner pixels inducing edge $(v_k, v_i)$ in $G_h$. By induction, there exists a path, say path $P$, of width $k$ from $x_k$ to $a2(x_k)$ that contains all the 1-pixels d- or hv-adjacent to hole $H_k$. Path $P$ is now used to show the existence of a path $P'$ of width $k$ from $x_i$ to $a2(x_i)$. Let $B_i$ (resp. $B_k$) be the 1-block of size $w(x_i)$ (resp. $w(x_k)$) associated with $x_i$ (resp. $x_k$). ¿From the construction of the hole graph we know $w(x_i) = w(x_k)$. It is possible to have $B_i = B_k$, $B_i$ overlapping with $B_k$, or, $B_i$ being disjoint, but adjacent to $B_k$ (this case applies to the situation shown in Figure 6). Imagine extending path $P$ into a path of width $k$ so that one end of the new path contains $x_i$ and $a1(x_i)$ and the other end contains $a2(x_i)$. Notice that no extension might be necessary on one end, as is the case in Figure 6 and that $x_i$ could be in both extensions. Let $P'$ be the so obtained path. Path $P'$ contains no 0-pixel. No pixel in

$B_i$ or $B_k$ can be a 0-pixel and the existence of a 0-pixel in $P'$ not contained in $B_i$ or $B_k$ would imply a contour pixel that cannot belong to a 1-block of size $k$. Hence, $P'$ represents a path of width $k$ from $a1(x_i)$ to $a2(x_i)$.

Let $v_j$ be the parent of vertex $v_i$ in the rooted tree and let $x'_i$ and $x_j$ be the two corner pixels inducing the edge $(v_i, v_j)$ in the hole graph. The situation when vertex $v_i$ is the root of the tree (i.e., no $v_j$ exists) is simplier and is omitted. 1-pixel $x'_i$ is the only corner pixel d-adjacent to hole $H_i$ for which we have not shown the existence of a path of with $k$ from $a1(x'_i)$ to $a2(x'_i)$. Using an argument identical to the one used for the leaves, the existence of a path of width $k$ from $x'_i$ to $a2(x'_i)$ containing all 1-pixels d- and hv-adjacent to hole $H_i$ is shown. The claim then follows.

It is easy to see that the existence of a path of width $k$ between any two 1-pixels hv- or d-adjacent to 0-pixels belonging to the same hole implies the existence of a path of width $k$ between any two 1-pixels in the image. From Properties 1 and 2 it then follows that $k = k^*$.

Assume now that the hole graph $G_h$ contains a cycle. Let $C$ be a cycle of length $l$ in $G_h$, $l \geq 1$. Every edge on cycle $C$ is induced by a unique pair of 1-pixels. Let $(v_i, v_j)$ be such an edge induced by the corner pixels $x_i$ and $x_j$. Any path from $a1(x_i)$ to $a2(x_i)$ must go through at least one of the bottlenecks inducing the edges of cycle $C$. In order to allow for such a path and be able to reach all 1-pixels in image $I$, we need $k^*$ to be at least as small as the largest cost associated with an edge on cycle $C$. Intuitively, we need to "open" the cycle. This is exactly what happens in the cycle-breaking algorithm in which the generated value $k'$ breaks every cycle in the hole graph. The existence of a path of width $k'$ in image $I$ follows from the above discussion. The lemma follows. □

## 4 The Mesh Algorithm

In this section we describe how to execute Stage 1 and Stage 2 on a mesh of size $n \times n$ in $O(n)$ time. Every 1-pixel of image $I$ can determine in $O(1)$ time what type of 1-pixel it is and what type of vertices it induces in the boundary graph. The weights of the vertices and the edges in the boundary graph can be determined in $O(n)$ time using straightforward data movement techniques. Hence, Stage 1 can be executed in $O(n)$ time.

In order to set up the hole graph needed in Stage 2, assume we have labeled the holes of image $I$. This can be done in $O(n)$ time using a connected component labeling algorithm [2, 6]. Using the weights of the boundary graph, the hole graph is then set up in $O(n)$ time. The remainder of this section describes how to determine the largest value $k'$ that breaks all cycles in hole graph $G_h$.

Hole graph $G_h$ can contain cycles of length 1 and 2 caused by self-loops and multiple edges. We handle such cycles first by performing local computations. If vertex $v_i$ of $G_h$ has a self-loop of cost $c$, we reduce $k$ to satisfy $k \leq c$ and delete the self-loop. If there exist two edges between vertices $v_i$ and $v_j$, one of cost $c_1$ and another of cost $c_2$, with $c_1 \leq c_2$, we remove the edge with cost $c_2$ and reduce $k$ to satisfy $k \leq \max\{c_1, c_2\}$.

Assume now that all self-loops and multiple edges have been removed from $G_h$. Let $k$ be the estimate of $k^*$ after the removal of these edges. Assume further that the updated graph $G_h$ contains no edges of cost $\geq k$. Our cycle-breaking algorithm uses binary search. Assume we are testing whether a given value $l$ breaks the cycles in $G_h$. We remove all edges of cost $\geq l$ and check whether the resulting graph is cycle-free (i.e., whether it is a forest). If it contains a cycle, $l$ is an upper bound for $k'$. If it contains no cycle, $l$ is a lower bound for $k'$ because it could be possible to break the cycles with a larger value. By choosing appropriate values for $l$ each time, we can determine the correct value of $k'$ in $O(\log |V_h|)$ iterations. In order to obtain an efficient parallel algorithm for the mesh architecture, we combine the binary search with a data reduction technique. More precisely, after one iteration we also reduce the size of the graph by at least one half. We next describe how to generate from $G_h$, after one cycle-breaking test, a graph of at most half the size.

Let $l$ be the median among the edge costs of graph $G_h$. Let $G_{h,l} = (V_{h,l}, E_{h,l})$ be the graph obtained from $G_h$ by deleting all edges of cost $\geq l$.

**Case 1:** $G_{h,l}$ contains a cycle.
In this case the algorithm continues with $G_{h,l}$. Observe that $|E_{h,l}| \leq |E_h|/2$.

**Case 2:** $G_{h,l}$ contains no cycles.
Graph $G_{h,l}$ consists of a collection of trees, $T_1, T_2, \ldots T_r$. Let $G'_{h,l} = (V'_{h,l}, E'_{h,l})$ be the graph used by the next iteration. $G'_{h,l}$ is generated as follows. We shrink every tree $T_i$ to a single vertex $u_i$ and let $V'_{h,l} = \{u_1, u_2, \ldots, u_r\}$. For vertex $v$ in $G_h$, let $t(v)$ be the tree containing vertex $v$. For every edge $(v_1, v_2)$ in $G_h$ having edge cost $c$ with $c \geq l$ we include in $G'_{h,l}$ the edge $(u_{t(v_1)}, u_{t(v_2)})$ with a cost of $c$. After all edges have been added, graph $G'_{h,l}$ contains self-loops and multiple edges. We remove self-loops and the multiple edges according to the rules stated above (doing so also improves our

estimate of $k'$).

Graph $G'_{h,l}$ may, however, not satisfy the requirement $|E'_{h,l}| \leq |E_h|/2$. When $G_h$ contains many edges of cost $l$, graph $G'_{h,l}$ may contain too many edges. In the extreme case, we can have $G'_{h,l} = G_h$ and we need to avoid an infinite loop. We proceed as follows. Assume $|E'_{h,l}| > |E_h|/2$. Let $\hat{l}$ be the smallest edge cost in $G_h$ with $\hat{l} > l$. If no such $\hat{l}$ exists, we have $k' = l$ and are done. Otherwise, let $G_{h,\hat{l}}$ be the graph obtained from $G_h$ by deleting all edges of cost $\geq \hat{l}$. When $G_{h,\hat{l}}$ contains a cycle, $k' = l$ and we are done. If $G_{h,\hat{l}}$ contains no cycle, we apply the shriking process described above to generate graph $G'_{h,\hat{l}}$. We now have a graph satisfying $|E'_{h,\hat{l}}| \leq |E_h|/2$ and the next iteration uses $G'_{h,\hat{l}}$.

Figure 7 illustrates one iteration of the cycle-breaking algorithm. Figure 7(a) shows an initial graph $G_h$, (b) shows $G_{h,3}$, (c) shows $G'_{h,3}$ and (d) shows $G'_{h,3}$ after self-loops and multiple edges have been removed. At this point we have $3 \leq k' \leq 6$. When continuing with $G'_{h,3}$, we do not need to consider $l = 3$ again (even though 3 is the median among the edge weights in $G'_{h,3}$). Using $l = 4$ does not break all cycles and thus the cycle-breaking algorithm returns $k' = 3$.

Assume every processor $i$ of the $n \times n$ mesh contains at most one edge $(u_i, u_j)$ of an at most $n^2$-vertex planar graph $G$. In order to complete the description of our algorithm we need to show that in $O(n)$ time we can determine the connected components of graph $G$ and can determine whether any of these components contains a cycle. We briefly sketch the main idea for an algorithm solving both problems. The algorithm uses a data reduction technique in which, in $O(n)$ time, the problem is reduced from one on at most $n^2$ vertices to one on at most $n^2/2$ vertices. We point out that the same time bounds have been claimed for general graphs in [13]. However, the algorithm we describe is simplier and makes use of the fact that we are dealing with planar graphs.

To start with, every vertex $u_i$ of $G$ selects, among the edges adjacent to it, the smallest-indexed vertex $u_i$ is incident to. The selected edges form a forest and we next use the algorithm described in [1] to determine the connected components of this forest. We then shrink each component representing a forest to a "supernode", put back the edges of $G$ not in the forest, and obtain a new graph $G'$. Every self-loop or pair of edges between the same pair of vertices in $G'$ represents a cycle in $G$. If we are testing for cycles in $G$, the existence of a self-loop or a multiple edge in $G'$ indicates the termination of the algorithm. In case we need to solve the connected component problem on $G$, we remove all self-loops and multiple edges in $G'$. It is easy to see that $G'$ contains at most $n^2/2$ vertices. Since $G$ is a planar graph, we have also reduced the number of edges by a constant fraction (this statement is not true for general graphs). We compress the remaining edges of $G'$ into the top-left corner of the mesh and recursively solve the connected component or the cycle testing problem on $G'$. Once the connected component numbers of the vertices in $G'$ are known, we can assign the correct component numbers to the vertices of $G$ in $O(n)$ time. The overall running time of the algorithm determining the connecting components or testing for the existence of cycles in thus $O(n)$.

Summarizing, we obtain the following result.

**Theorem 3** *Given an image $I$ stored in an $n \times n$ mesh of processors with* *one pixel per processor, the largest $k$ such that $I$ is $k$-width-connected can be determined in $O(n)$ time.*

## 5 Conclusion

In this paper we considered the problem of determining, for a binary image $I$ stored in a $n \times n$ mesh of processors, the largest integer $k$ such that $I$ is k-width-connected. We present an optimal $O(n)$ time solution to this problem. By having every pixel (i.e., the respective processor) perform local computations, our algorithm generates first a preliminary estimate of the result. The final result is then obtained by generating a graph that models bottlenecks in image $I$ and applying a cycle-breaking algorithm to this graph.

## 6 Acknowledgements

We thank the referees for constructive comments and for identifying a flaw in an earlier definition of k-width-connectivity.

## References

[1] Atallah, M.J., Hambrusch, S.E., "Solving Tree Problems on a Mesh-connected Processor Array", *Information and Control*, Vol. 69, pp. 168-187, 1986.

[2] Cypher, R., Sanz, J., Snyder, L., "Algorithms for Image Component Labeling on SIMD Mesh Connected Computers", *IEEE Trans. on Computers*, 1990, Vol. 39, pp. 276-281.

[3] Cypher, R., Sanz, J., Snyder, L., "Hypercube and shuffle- exchange algorithms for image component labeling", *Journal of Algorithms*, 1989, Vol. 10, pp. 140-150.

[4] Dehne, F., Hambrusch, S.E., "Parallel Algorithms for Determining k-Width-Connectivity in Binary Images", *JPDC*, 1990, Vol. 12, Nr. 1, pp. 12-23.

[5] Dyer, C., Rosenfeld, A., "Parallel Image Processing by Memory- Augmented Celluar Automata", *IEEE Pami*, 1981, Vol. 3, pp 29- 41.

[6] Hambrusch, S.E., TeWinkel, L., "A Study of Connected Component Algorithms on the MPP", *Proc. of 3rd Internat. Conf. on Supercomputing*, pp 477-483, 1988.

[7] Lim, W., Agrawal, A., Nekludova, L., "A fast parallel algorithm for labeling connected components in image arrays", *Parallel Processing for Computer and Vision Display*, Ed. P. Dew, R. Earnshaw, and T. Heywood, Addison-Wesley, pp 169-179, 1989.

[8] Levialdi, S., "On Shrinking Binary Picture Patterns", *CACM*, 1972, Vol. 15, pp. 7-10.

[9] Mead, C.A., Conway, L.A., *Introduction to VLSI Systems*, Addison Wesley, 1980.

[10] Miller, R., Stout, Q., *Parallel Algorithms for Regular Architectures*, manuscript, to be published by MIT press.

[11] Nassimi, D., Sahni, S., "Finding Connected Components and Connected Ones on a Mesh-Connected Parallel Computer", *SIAM J. on Comp.*, 1980, pp. 744-757.

[12] Pavlidis, T., *Algorithms for Graphics and Image Processing*, CSP, 1982.

[13] Reif, J., Stout,Q., "Optimal Component Labeling Algorithms for Mesh Computers and VLSI", manuscript.

[14] Rosenfeld, A., "Connectivity in Digital Pictures", *JACM*, 1970, Vol. 17, pp. 146-160.

Figure 1: A path of width 3 between 1-pixels $a$ and $b$

Figure 2: An image that is 3-width, but not 4-width connected

Figure 3: The boundary graph of the image shown in Figure 2

Figure 4: Illustrating Lemma 1 with $w(x) = 5$, $w(x, y) = 3$, 0-pixel $p$ makes $w(y, t) > w(y, x)$ impossible

(a) assume $k = 5$ from Stage 1

(b) edges between $H_1$ and $H_2$ induced by $x$, $x'$, and $x''$

Figure 5: Creating edges of the hole graph

$k = 6$, $w(x_i) = w(x_k) = 4$, dashed lines show extension of $P$ into $P'$

Figure 6: Extending path $P$ into path $P'$

(a) Graph $G_h$; edges included in $G_{h,3}$ are in bold

(b) $G'_{h,3}$ with self-loops and multiple edges

(c) $G'_{h,3}$ without self-loops and multiple edges; at this point we have $k' \leq 6$

Figure 7: Breaking cycles