Parallel Computing 12 (1989) 221–231
North-Holland

Computing the configuration space for a robot on a mesh-of-processors *

Frank DEHNE, Anne-Lise HASSENKLOVER and Jörg-Rüdiger SACK

Center for Parallel and Distributed Computing, School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6

Received June 1988

Abstract. In this paper, we present a systolic algorithm for computing the configuration space of an arrangement of arbitrary obstacles in the plane for a rectilinearly convex robot. The obstacles and the robot are assumed to be represented in digitized form by a $\sqrt{n} \times \sqrt{n}$ binary image. The algorithm is designed for a Mesh-of-Processors architecture with n processors (using the canonical representation of an image on a processor array) and has an execution time of $O(\sqrt{n})$ which is asymptotically optimal.

Keywords. Computational geometry, robotics, image processing, systolic algorithms.

1. Introduction

1.1. Problem description

Recently, a growing interest in motion planning problems has been observed. This is due to the variety of application areas in which such problems arise: e.g., robotics and computer graphics.

The classical path planning problem is to find a shortest path for a robot, R, located at some initial position to some final position in the presence of a collection of polygonal obstacles. During the entire motion, no collision may occur between the robot and any of the obstacles.

The problem is easily solved if the robot is representable by a single point, p. In this case, the solution reduces to finding a shortest path in a graph, for which any of the known shortest-path graph algorithms can be employed. To construct this graph, first compute the visibility graph from p with respect to the vertices of all obstacles and the final robot position; then, this visibility graph is repeatedly augmented by computing those new vertices that are visible from a vertex reached previously.

While it is simple to solve the problem for point-robots, this does not seem to be a very realistic assumption. A more realistic view was taken by Lozano-Perez who assumed the robot to be representable by a convex polygon. In [7], he developed a technique known as the *configuration space* approach useful to solve these more general robot problems.

The idea is to shrink the robot, R, to a single point, r, which is referred to as the *reference point* of R. Then, the obstacles are grown in such a way that a shortest path for r in the presence of the grown obstacles is a shortest path for R in the presence of the original

* The first and third author's research was supported by the Natural Sciences and Engineering Research Council of Canada.

0167-8191/89/$3.50 © 1989, Elsevier Science Publishers B.V. (North-Holland)

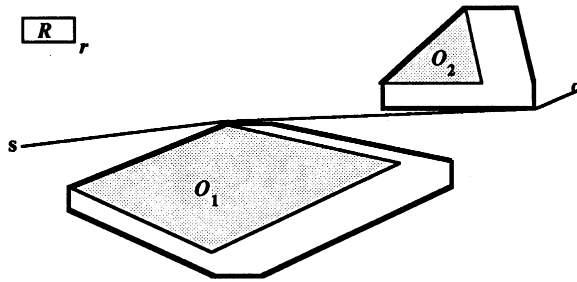

Fig. 1. The configuration space of polygonal figures.

obstacles. The growing operation can be visualized by placing a pen at the reference point of the robot and sliding the robot around the obstacles while keeping the same orientation. The lines traced by the pen determine the grown obstacles. See Fig. 1.

Formally, the configuration space of a set $\mathcal{O} = \{O_1, \ldots, O_m\}$ of m obstacles for a robot R with reference point $r \in R$ is the set

$$C(\mathcal{O}, R) := \bigcup_{1 \leqslant i \leqslant m} C(O_i, R);$$

i.e., the union of the *grown obstacles* $C(O_i, R)$ defined as follows: Let $R' := \{x - r \mid x \in R\}$ denote the *inverted robot* R (with respect to r), then

$$C(O_i, R) := \bigcup_{p \in O_i} C(p, R'), \quad \text{where } C(p, R') := \{p + q \mid q \in R'\}.$$

In addition to the above-mentioned path finding problem, the configuration space has proved to be a valuable tool for a number of other related motion problems (see, e.g., [13]).

For robotics applications, real-time constraints frequently apply. Using standard sequential computers it may be impossible to meet these time constraints. This motivated our research in computing the configuration space on a parallel computer architecture.

The parallel machine we selected is the Mesh-of-Processors architecture which will be briefly reviewed in the following Section 1.2. The architecture has been built and is already used in many applications such as image processing and computer graphics. (For a survey on computational geometry algorithms on this and other parallel machines the reader is referred to the survey article [4].)

As we will show in the remainder of this paper, our approach via the Mesh-of-Processors has two advantages:

(a) it yields a significant speedup of the computation time for constructing the configuration space, and

(b) it allows the configuration space to be computed directly from the image of the arrangement of obstacles (and the robot); e.g., a photograph or sensor data. The algorithm in [7] mentioned above assumes the obstacles to be approximated by polygons; for image data (which is the usual form of input in practice) the polygonal approximation would have to be computed in an additional preprocessing step.

1.2. The Mesh-of-Processors architecture

Two-dimensional arrays of processors have long been proposed for image processing because $\sqrt{n} \times \sqrt{n}$ binary images can be naturally mapped onto a Mesh-of-Processors of size n which is an arrangement of n processing elements PE(i, j) arranged on a square grid. Each PE(i, j) is connected by bi-directional unit-time communication links to each of its four direct neighbors

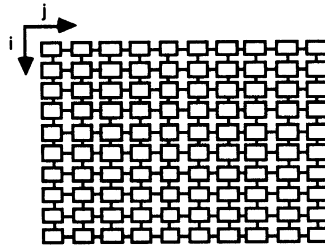

Fig. 2. A Mesh-of-Processors of size 100.

(if these exist) as shown in Fig. 2. Furthermore, it is usually assumed that every PE has only a constant amount of memory cells referred to as *registers*.

In image processing, processor arrays are mostly used for low-level local operations such as image restoration, noise removal, computation of connected components, and edge detection (cf. [2,6,11,12,15]). Recently, they have also been proposed as a machine model for Computational Geometry. Miller and Stout [8,10] have presented $O(\sqrt{n})$ time algorithms for computing the distance between two images, determining extremal points, diameter, and smallest enclosing circle of an image, as well as for testing convexity and separability of digitized images. Dehne et al. [5] have proposed $O(\sqrt{n})$ algorithms for computing the contours of an image, and all kth rectilinear convex hulls of an image stored on a mesh. In [3], Dehne et al. presented an optimal algorithm for computing parallel visibility for an image stored on a mesh.

1.3. Computing the configuration space on a Mesh-of-Processors

In the following Sections 2 and 3, we will study the problem of computing the configuration space of an arrangement $\mathcal{O}$ of arbitrary polygonal obstacles for a rectilinearly convex robot R. A polygon is *rectilinearly convex* if the intersection of the robot with any horizontal or vertical line consists of at most one interval. The obstacles and the robot are represented as a digitized $\sqrt{n} \times \sqrt{n}$ image on a Mesh-of-Processors of size n using the canonical representation; see Fig. 3 for an illustration. Each pixel contained in an obstacle will be called a *black* pixel while all other pixels will be called *white*.

In the remainder, we will refer to an obstacle (or robot) in image representation as *digitized* obstacle (or robot, respectively). For images, the same definition of the configuration space $C(\mathcal{O}, R)$ as given in Section 1.1 applies with $\mathcal{O}$ and R being sets of integer lattice points instead of arbitrary planar point sets.

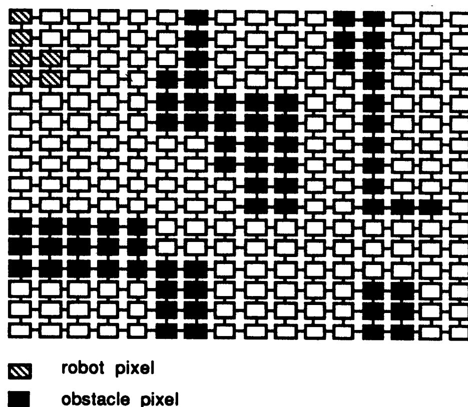

Fig. 3. Representing the robot and the obstacles on a Mesh-of-Processors.

The configuration space $C(\mathcal{O}, R)$ will be reported by the Mesh-of-Processors by coloring all pixels (i.e., lattice points) in $C(\mathcal{O}, R)$ black.

Assume, for a moment, that the robot has a boundary description which requires only a constant amount of space. In this case, the configuration space $C(\mathcal{O}, R)$ can be computed in a straightforward manner: a description of R' is broadcast to all pixels of the obstacles; every PE representing a black pixel p will then trace the robot boundary and subsequently set all pixels in $C(p, R')$ to black. This solution would require $O(\sqrt{n})$ time.

In general, the description of the robot is not of constant size but may be as large as $O(\sqrt{n})$ [using an optimal representation of rectilinearly convex robots; if we store the individual pixels the size may even be $\Omega(n)$]. Therefore, two problems arise: each processor cannot store a complete description of the robot, and, if each processor representing a black pixel p attempts to send a message to all pixels in $C(p, R')$ to color those pixels black, congestion of communication lines may occur for neighboring black pixels due to the massive amount of messages generated.

Storage size and congestion are the main problems to be solved in the remainder of this paper. We will present an $O(\sqrt{n})$ time algorithm for computing, for a rectilinearly convex digitized robot, the configuration space of an arbitrary arrangement of digitized obstacles. The running time of the algorithm is $O(\sqrt{n})$ and is, therefore, asymptotically optimal for the Mesh-of-Processors of size n. Furthermore, we will show that our algorithm also performs correctly and with the same time complexity for a more general type of robot.

Note that, from this algorithm an optimal parallel solution to the problem of finding a shortest path follows by applying the $O(\sqrt{n})$ time closest internal distance algorithm presented in [9].

The first step of our algorithm is to encode the robot appropriately. Then we will show how the configuration space can be computed in optimal time. We use a particular pipelining technique to communicate R' to all black pixels and combine this with concurrent pipelining processes (one for each black pixel) that color for each black pixel p all pixels in $C(p, R')$. It must be ensured that during the entire process no pixel receives more than a constant number of messages and that the amount of storage needed by each processing element is bounded by a constant.

2. The robot encoding

To communicate the shape of the robot to all obstacles, the shape needs to be properly encoded. The encoding is done as a preprocessing step to the actual algorithm and will be discussed in this section. For the remainder we will assume w.l.o.g., that the robot's reference point, r, is the leftmost pixel in the bottom-most column of R' (as defined above).

First, the inverted robot, R', of robot R is computed as follows:

- The coordinates of the reference point r are broadcast to all pixels of R.
- Each pixel of R computes the coordinates of the corresponding pixel in R'.
- Using a RAW (Random Access Write) procedure, as described in [8], all pixels of R are moved to the coordinates of the corresponding pixel in R'. (An appropriate translation of R will ensure that R' is completely contained in the $\sqrt{n} \times \sqrt{n}$ image.)

We define the *left boundary chain* of R' as follows: Consider the reference point r and the rightmost pixel t in the topmost column of pixels of R'; a pixel p of R' is called a *left boundary pixel* of R' if and only if the lower-left, left, upper-left, or upper-right neighbor of p is not contained in R'. The left boundary chain of R' is the shortest path from r to t among all paths visiting all left boundary pixels of R' and connecting two pixels only if they are

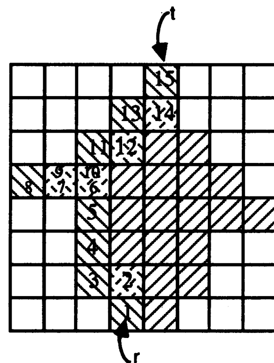

Fig. 4. The left boundary chain (pixel 1 through 15) of a rectilinearly robot's inverted image R'.

four-neighbors; see Fig. 4. Notice that some pixels may be visited twice, in Fig. 4 these are the pixels labelled 6 (10) and 7 (9).

R' is encoded by a sequence of *left boundary records*, one record for each occurrence of a left boundary pixel in the left boundary chain. Each left boundary record is a triple

$$\langle \text{RecordNo}, \text{NoOfPixelsToTheRight}, \text{DirectionOfNextPixel} \rangle$$

where RecordNo denotes the rank of a respective left boundary pixel in the left boundary chain, NoOfPixelsToTheRight is the number of pixels in R' to the right of the respective pixel, and DirectionOfNextPixel is the direction (given as left, up, or right) from the respective pixel to its immediate successor in the left boundary chain.

An example illustrating this encoding scheme is given in Fig. 5. Notice that the reference point r is always represented by the first boundary record. The entire left boundary chain can be traced by starting at the first boundary record and following, at each step, the direction specified by the DirectionOfNextPixel-field. Since a given left boundary pixel may occur twice in the left boundary chain it will, in this case, also be represented by two different left boundary records.

The encoding of R' can be computed in time $O(\sqrt{n})$ as follows:

- Each pixel of R', in parallel, determines whether it is a left boundary pixel and whether it will appear in one or two left boundary records.
- In parallel, each left boundary pixel determines its left boundary records (at most two). For each such record, the pixel's successor in the left boundary chain as well as the number of pixels to its right is determined.

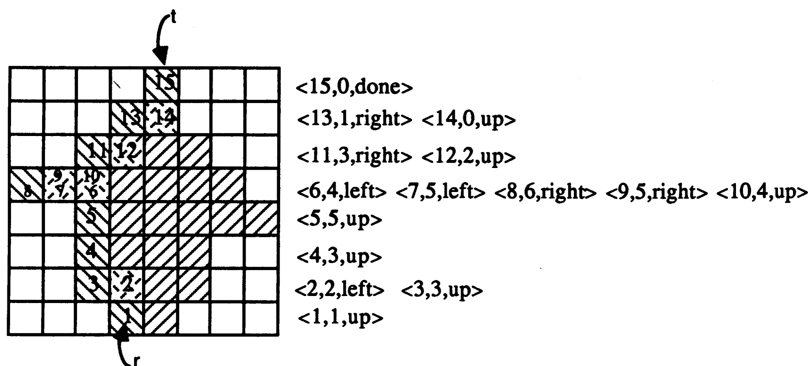

Fig. 5. Encoding of a rectilinearly convex robot's inverted image R'.

- For each record, the value of the field RecordNo is the rank of the record in the left boundary chain. Using the list ranking algorithm described in [1], all values RecordNo can be computed in $O(\sqrt{n})$ steps.

The time complexity of each step, and thus the total preprocessing time, is $O(\sqrt{n})$.

3. The pipelining algorithm

We assume that the encoding of R' has been determined as described in Section 2. We now turn our attention to the actual computation of the configuration space of the digitized obstacles stored on the Mesh-of-Processors. The basic idea is to send a description of R' to every black pixel p. Then, each such p colors black all pixels contained in $C(p, R)$.

When implementing this approach on a mesh, the following problems have to be solved:

- Each PE is limited to a constant amount of memory. Therefore, it cannot store the entire description of R' at any point in time.
- To color all pixels in $C(p, R')$ black, each black pixel p has to send a message to the pixels in $C(p, R')$. This may create congestion problems between messages of neighboring black pixels. In addition to the coloring messages, messages are sent to each black pixel to communicate the encoding of R'. These messages must not interfere with the coloring messages.

The solution to these problems is to pipeline the encoding of the robot in a particular fashion. As soon as a black pixel receives the first left boundary record of R', it starts a boundary-tracing process which colors all pixels in $C(p, R')$. These coloring processes also operate in a pipelined fashion; proper sequencing ensures that no congestion occurs.

Consider a leftmost pixel l of R'; clearly, l is a left boundary pixel. Its corresponding left boundary record, with field RecordNo $= k_1$, splits the sequence of left boundary records into two subsequences, called L_1 and L_2, of those records whose RecordNo-fields are smaller (or equal) and larger than k_1, respectively.

L_1 and L_2 are determined in time $O(\sqrt{n})$ by broadcasting k_1 to all boundary pixels. Then, each subsequence is sorted in snake-like ordering as indicated in Fig. 6 using, e.g., the sorting technique described in [14].

The configuration space is then computed in three phases:

(1) The sequence L_1 of boundary records is pipelined through the entire mesh.

(2) The sequence L_2 of boundary records is pipelined through the entire mesh.

(3) All pixels contained in $C(\emptyset, R)$ (contained in the image) are colored black.

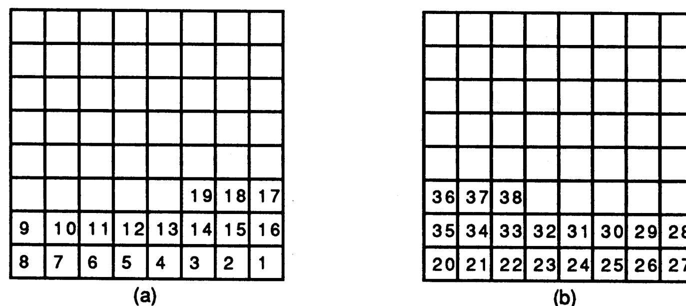

Fig. 6. Storage schemes for L_1 (a) and L_2 (b). The numbers represent the field *RecordNo* of the left boundary records; $k_1 = 19$.

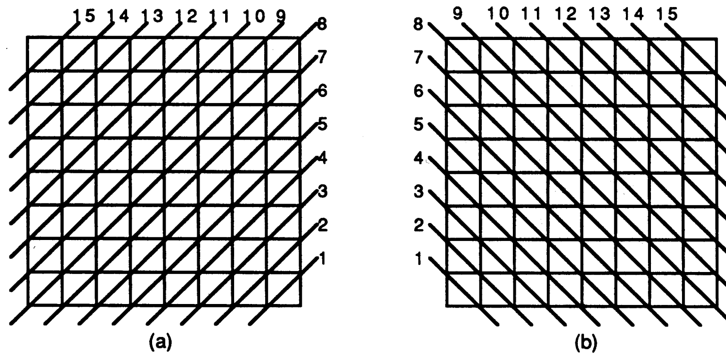

Fig. 7. Pipelining schemes for L_1 (a) and L_2 (b). The numbers represent the sequencing for the pipelining of each left boundary record through the diagonals of pixels.

These three phases will now be described in detail. We assume that each processor PE(i, j) has three additional registers (which are initialized to zero):

RN1, RN2: Integer registers for storing the RecordNo-value of the next two boundary records to be processed.

PR: An integer register for storing the number of pixels to the right of pixel (i, j) which will be subsequently colored black in Phase 3.

Let $lb_1, \ldots, lb_{k_1}$ and $lb_{k_1+1}, \ldots, lb_{k_2}$ be the sequences L_1 and L_2 of left boundary records, respectively, stored in snake-like ordering as described above. The idea of Phase 1 and Phase 2 is to pipeline L_1 and L_2 through all processors in the fashion indicated in Fig. 7(a), (b), respectively. During these processes each PE computes its value PR. Each *line* indicates those processors that are reached by a common boundary record at one point in time. More precisely: first, lb_1 is sent to the processor on line 1; then, lb_1 advances from line 1 to line 2, and, simultaneously, lb_2 is sent to the processor on line 1; then, lb_1 advances to line 3, lb_2 advances to line 2, and lb_3 is sent to line 1; etc.

The motion of these lines corresponds to a motion of waves. The directions in which the waves propagate for the sequences L_1 and L_2 are different. This is crucial to ensure that the pipelining process of the robot encoding does not interfere with the messages created to color, for each black pixel p, all pixels in $C(p, R')$.

Each time a black pixel is reached in Phase 1 by lb_1, a message will be sent in the direction (left or up) indicated by the field DirectionOfNextPixel at lb_1, to trace the left boundary of R' with its reference point r located at the particular pixel. Subsequent left bqundary records lb_2,.... will be forwarded on this path such that each pixel on the left boundary of $C(p, R')$ will receive the respective left boundary record and the value NoOfPixelsToTheRight. Obviously, it may receive several such values from several neighboring black pixels in which case it stores the maximum, only.

The following is the detailed description of Phase 1. Note that the algorithm assumes the execution of all PEs to be synchronized; i.e., all processors are always in the same stage of the algorithm.

FOR $i = 1$ TO $k_1 + \sqrt{n}$ DO

(a) IF $i \leqslant k_1$

THEN lb_i is sent to PE($\sqrt{n}, \sqrt{n}$). (In fact, lb_i is always stored at PE($\sqrt{n}, \sqrt{n}$).) All subsequent other left boundary records of L_1 advance one position in the snake like ordering described above.

(b) Each processor PE(i, j) that receives a left boundary record
⟨RecordNo, NoOfPixelsToTheRight, DirectionOfNextPixel⟩
performs:
Condition1 := (RecordNo = 1) AND (PE(i, j) represents a black pixel);
Condition2 := (RN1 = RecordNo) OR (RN2 = RecordNo);
IF Condition1 OR Condition2
THEN set PR := max{PR,NoOfPixelsToTheRight} and send a record number RecordNo + 1 to the processor indicated by DirectionOfNextPixel (if exists).
(c) Each processor PE(i, j) that receives a record number RN, sets RN1 := RN2 and RN2 := RN.
Note that, if a processor receives more than one record number, then these record numbers are identical; see Lemma 2 in Section 4.
(d) Each processor PE(i, j) that received, in Step b, a left boundary record forwards this record to processors PE(i, $j-1$) and PE($i-1$, j), if exist.

Phase 2 is analogous to Phase 1; the only differences are that the FOR loop is iterated from k_1+1 to $k_2+\sqrt{n}$ and, in Step (d), the received boundary record is forwarded to PE(i, $j+1$) and PE($i-1$, j). In step (b), Condition1 will never be true during Phase 2 and may, therefore, be removed.

In the following Section 4, we will show that upon completion of Phase 2, each processor will have stored in its register PR the number of pixels to its right which need to be colored black in order to compute the configuration space C($\mathcal{O}$, R).

The implementation of the final Phase 3 is now quite straightforward:
- Every processor whose PR register is not equal to 0 changes the color of the pixel it represents to black and sends a message ⟨PR⟩ to its right neighbor.
- Then, the following process is iterated $\sqrt{n}$ times: each processor receiving a message ⟨SomeNumber⟩ from its left neighbor sets the color of its pixel to black. If SomeNumber is greater than 1, its sends the message ⟨SomeNumber − 1⟩ to its right neighbor (if exists).

4. Correctness and time complexity of the algorithm

In this section, we will establish the correctness of the above algorithm. The interesting part is to show that after Phase 1 and Phase 2 have been completed, the PR registers of all pixels contain the correct values. More specifically, each pixel contains in its PR register the number of black pixels located to its right that have to be colored black in order to obtain the configuration space.

For any black pixel p, let $L(p) := (p = p_1, p_2, p_3, \ldots, p_{k_2})$ denote the left boundary chain of R' when translated such that r coincides with p; $L_1(p) := (p_1, p_2, p_3, \ldots, p_{k_1})$ and $L_2(p) := (p_{k_1+1}, p_2, p_3, \ldots, P_{k_2})$ denote the translated boundary records of L_1 and L_2, respectively.

For the remainder, one time unit will refer to the time necessary for one execution of the loop body.

If, during Phase 1, a black pixel p receives lb_1 (i.e., Condition1 = true), it updates its register PR and initiates a sequence of record numbers to trace $L(p)$. That is, in Phase 1, p sends a record number '2' to its successor p_2 in $L_1(p)$. Pixel p_2 will store this record number for two time units using its registers RN1 and RN2 (simulating a queue of length 2). During this time, it will be 'sensitive' (not only to receiving lb_1, but also) to receiving a left boundary record with RecordNo = 2; that is, when receiving such a record, Condition2 will be true. The processor will update its register PR and send a record number '3' to p_3, etc.

Lemma 1. *If during Phase* 1 *a black pixel* p *receives* lb_1 *at time* t_1, *then every* $p_i \in L_1(p)$, $i \geq 2$, *receives record number* 'i' *at time* $t_i = t_1 + 2(i-2)$ *and* lb_i *at time* $t_i' = t_1 + 2(i-1)$.

Proof. By induction on $i \in \{2, \ldots, k_1\}$. When p_1 receives lb_1, it sends a record number '2' to p_2 which receives this message during the same time unit; i.e., at time $t_2 = t_1$. In the pipelining process for boundary records, lb_2 will be received by p_2 two time units later, i.e., at time $t_2' = t_1 + 2$. Assume that Lemma 1 holds for all $p_2, \ldots, p_i$ for some $2 \leq i < k_1$. Hence, at time $t_i = t_1 + 2(i-2)$, p_i receives a record number 'i' and p_{i-1} receives lb_{i-1}. Since p_{i-1}, p_i, and p_{i+1} are subsequent pixels in $L_1(p)$, it follows from the definition of the left boundary chain of R' that p_i is either an upper or a left neighbor of p_{i-1}, and p_{i+1} is either an upper or a left neighbor of p_i. Therefore, p_{i-1}, p_i, and p_{i+1} belong to subsequent lines of processors with respect to the pipelining scheme for boundary records (recall Fig. 7(a)). Hence, p_{i-1} receives lb_i at time $t_i + 1$ and then, at time $t_i + 2$, it receives lb_{i+1}. Subsequently, lb_{i+1} is received by p_i at time $t_i + 3$ and, therefore, by $p_i + 1$ at time $t_i + 4 = t_1 + 2i = t_{i+1'}$. Furthermore, since p_{i-1} receives lb_i at time $t_i + 1$, p_i receives lb_i at time $t_i + 2$. Since, by assumption, p_i has received record number 'i' at time t_i, it follows for the execution of Step (b) during time unit $t_i + 2$, that p_i's register RN2 contains the value i. Summarizing, at time $t_i + 2$, p_i receives lb_i and has Condition2 = true; hence, p_i sends a record number "$i + 1$" to p_{i+1} which will receive it during the same time unit, i.e., at time $t_i + 2 = t_1 + 2(i-1) = t_{i+1}$. □

Analogously, it can be shown that in Phase 2, the pixels $p_{k_1+1}, \ldots, p_{k_2}$ receive and process the record numbers $k_1 + 1, \ldots, k_2$ and the boundary records $lb_{k_1+1}, \ldots, lb_{k_2}$, respectively, in the same manner.

Summarizing, for every black pixel that receives lb_1, a trace of $L(p)$ is initiated where every pixel p_i receives lb_i and updates its register PR such that upon completion of this process, each $p_i \in L(p)$ contains in its register PR the number of black pixels to its right in $C(p, R')$.

The pipelining process for the boundary records ensures that every pixel receives a left boundary record lb_1. Hence, the above tracing process is started for each black pixel p. Every pixel stores in its register PR the maximum PR-value for all tracing processes in which it is involved (see Step (b) of the loop body).

In order to prove the correctness of the algorithm, it remains to be shown that these tracing processes do not interfere with each other or with the pipelining process for the left boundary records.

Interference between tracing processes can only occur if two such processes attempt to send two different record numbers to the same pixel at the same time. However, this can not happen because of the following lemma.

Lemma 2. *If a processor receives two record numbers at the same time, then these record numbers are identical.*

Proof. During Phase 1, all record numbers are either sent upwards or to the left. Therefore, if a processor receives two record numbers, these must have originated from the right and from below, respectively. These two neighboring processors, however, are on the same line with respect to the pipelining process for boundary records (see Fig. 7(a)). Hence, they must have received the same boundary record, and, therefore, they must have sent the same record number. A similar argument holds for Phase 2. □

Obviously, there is no interference between tracing processes and the pipelining process for boundary records since, for each execution of the loop body, record numbers are sent in Step (b) and boundary records in Steps (a) and (d), only.

We therefore obtain,

Lemma 1. *If during Phase* 1 *a black pixel p receives* lb_1 *at time* t_1, *then every* $p_i \in L_1(p)$, $i \geq 2$, *receives record number* 'i' *at time* $t_i = t_1 + 2(i-2)$ *and* lb_i *at time* $t_i' = t_1 + 2(i-1)$.

Proof. By induction on $i \in \{2, \ldots, k_1\}$. When p_1 receives lb_1, it sends a record number '2' to p_2 which receives this message during the same time unit; i.e., at time $t_2 = t_1$. In the pipelining process for boundary records, lb_2 will be received by p_2 two time units later, i.e., at time $t_2' = t_1 + 2$. Assume that Lemma 1 holds for all $p_2, \ldots, p_i$ for some $2 \leq i < k_1$. Hence, at time $t_i = t_1 + 2(i-2)$, p_i receives a record number 'i' and p_{i-1} receives lb_{i-1}. Since p_{i-1}, p_i, and p_{i+1} are subsequent pixels in $L_1(p)$, it follows from the definition of the left boundary chain of R' that p_i is either an upper or a left neighbor of p_{i-1}, and p_{i+1} is either an upper or a left neighbor of p_i. Therefore, p_{i-1}, p_i, and p_{i+1} belong to subsequent lines of processors with respect to the pipelining scheme for boundary records (recall Fig. 7(a)). Hence, p_{i-1} receives lb_i at time $t_i + 1$ and then, at time $t_i + 2$, it receives lb_{i+1}. Subsequently, lb_{i+1} is received by p_i at time $t_i + 3$ and, therefore, by $p_i + 1$ at time $t_i + 4 = t_1 + 2i = t_{i+1'}$. Furthermore, since p_{i-1} receives lb_i at time $t_i + 1$, p_i receives lb_i at time $t_i + 2$. Since, by assumption, p_i has received record number 'i' at time t_i, it follows for the execution of Step (b) during time unit $t_i + 2$, that p_i's register RN2 contains the value i. Summarizing, at time $t_i + 2$, p_i receives lb_i and has Condition2 = true; hence, p_i sends a record number "$i+1$" to p_{i+1} which will receive it during the same time unit, i.e., at time $t_i + 2 = t_1 + 2(i-1) = t_{i+1}$. □

Analogously, it can be shown that in Phase 2, the pixels $p_{k_1+1}, \ldots, p_{k_2}$ receive and process the record numbers $k_1 + 1, \ldots, k_2$ and the boundary records $lb_{k_1+1}, \ldots, lb_{k_2}$, respectively, in the same manner.

Summarizing, for every black pixel that receives lb_1, a trace of $L(p)$ is initiated where every pixel p_i receives lb_i and updates its register PR such that upon completion of this process, each $p_i \in L(p)$ contains in its register PR the number of black pixels to its right in $C(p, R')$.

The pipelining process for the boundary records ensures that every pixel receives a left boundary record lb_1. Hence, the above tracing process is started for each black pixel p. Every pixel stores in its register PR the maximum PR-value for all tracing processes in which it is involved (see Step (b) of the loop body).

In order to prove the correctness of the algorithm, it remains to be shown that these tracing processes do not interfere with each other or with the pipelining process for the left boundary records.

Interference between tracing processes can only occur if two such processes attempt to send two different record numbers to the same pixel at the same time. However, this can not happen because of the following lemma.

Lemma 2. *If a processor receives two record numbers at the same time, then these record numbers are identical.*

Proof. During Phase 1, all record numbers are either sent upwards or to the left. Therefore, if a processor receives two record numbers, these must have originated from the right and from below, respectively. These two neighboring processors, however, are on the same line with respect to the pipelining process for boundary records (see Fig. 7(a)). Hence, they must have received the same boundary record, and, therefore, they must have sent the same record number. A similar argument holds for Phase 2. □

Obviously, there is no interference between tracing processes and the pipelining process for boundary records since, for each execution of the loop body, record numbers are sent in Step (b) and boundary records in Steps (a) and (d), only.

We therefore obtain,

Lemma 3. *After Phases* 1 *and* 2 *have been completed, for each pixel* p_i *the registers PR contain the number of pixels to right of* p_i *which, if colored black, correctly produce* $C(\mathcal{O}, R)$.

Phase 3 of the algorithm simply performs this coloring of black pixels and thus the algorithm correctly determines $C(\mathcal{O}, R)$.

In order to determine the time complexity of the algorithm, we observe that all preprocessing steps can be executed in time $O(\sqrt{n})$. The number, k_2, of left boundary records of R' is bounded by the sum of the number of rows and the number of columns of R'. Thus it follows that $k_2 = O(\sqrt{n})$ implying that Phases 1 and 2 of the algorithm execute the body of the FOR loop at most $O(\sqrt{n})$ times. Each execution of the loop body takes O(1) time; hence, Phase 1 and 2 have a time complexity of $O(\sqrt{n})$. It is easy to see that Phase 3 takes $O(\sqrt{n})$ time. Therefore, the total time complexity of the algorithm is $O(\sqrt{n})$. Furthermore, only three additional registers are required by each processor.

Summarizing, we obtain

Theorem 1. *The configuration space* $C(\mathcal{O}, R)$ *of an arbitrary set* $\mathcal{O}$ *of digitized obstacles for a rectilinearly convex digitized robot can be computed on a Mesh-of-Processors of size n in time* $O(\sqrt{n})$ *which is asymptotically optimal.*

5. Extensions and open problems

The type of robot considered in the previous sections, the rectilinearly convex robot, is already slightly more general than the convex robot type studied by Lozano-Perez [7] in the sequential model of computation. However, a further minor generalization is easily obtained. Let R be a horizontally convex robot (i.e., the intersection with every horizontal line consists of at most one line segment) which can be partitioned (by horizontal lines) into a constant number of rectilinearly convex pieces. By partitioning R into these pieces and applying Phases 1 and 2 of the algorithm to the sequence of left boundary records for each piece, one after the other in bottom up order, an optimal algorithm for computing the digitized configuration space is obtained.

Whether it is possible to further generalize the type of robot while maintaining the algorithm's optimal time performance is posed as an open problem.

References

[1] Attallah, M.J. and S.E. Hambrusch, Solving tree problems on a mesh-connected processor array, in: *Proc. 26th Symp. on Found. of Comput. Sci* (1985) 222–231.

[2] P.E. Danielson and S. Levialdi, Computer architecture for pictorial information systems, *IEEE Comput.* (November 1981).

[3] F. Dehne, A.-L. Hassenklover, J.-R. Sack and N. Santoro, Parallel visibility on a mesh-connected parallel computer, in: *Proc. Int. Conference on Parallel Processing and Applications,* L'Aquila, Italy (1987) 173–180.

[4] F. Dehne and J.-R. Sack, A survey of parallel computational geometry algorithms, in: *Proc. 5th Int. Workshop on Parallel Processing by Cellular Automata and Arrays*, Berlin, GDR (1988).

[5] F. Dehne, J.-R. Sack and N. Santoro, Computing on a systolic screen: hulls, contours and applications, in: *Proc. Conf. Parallel Architectures and Languages*, Eindhoven, The Netherlands (1987) 121–133.

[6] R.A. Klette, Parallel computer for image processing, *Elektron. Inform. Kyber.* **15** (5/6) (1979) 237–263.

[7] T. Lozano-Perez, Spatial planning: A configuration space approach, *IEEE Trans. Comput.* **32** (2) (1983) 108–120.

[8] R. Miller and Q.F. Stout, Computational geometry on a mesh-connected computer, in: *Proc. Int. Conf. on Parallel Processing* (1984) 66–73.

[9] R. Miller and Q.F. Stout, Mesh computer algorithms for line segments and simple polygons, in: *Proc. Int. Conf. on Parallel Processing* (1987) 282–285.
[10] Q.F. Stout and R. Miller, Mesh-connected computer algorithms for determining geometric properties of figures, in: *Proc. 7th Int. Conf. on Pattern Recognition*, Montréal, Canada (1984).
[11] D. Nassimi and S. Sahni, Finding connected components and connected ones on a mesh-connected parallel computer, *SIAM J. Comput.* **9** (4) (1980).
[12] A.P. Reeves, Survey, parallel computer architectures for image processing, *Comput. Vision Graph. Image Process.* **25** (1984).
[13] M Sharir, R. Cole, K. Kedem, D. Leven, R. Pollack and S. Sifrony, Geometric applications of Davenport–Schinzel sequences, in: *Proc. 27th Symp. on Found. of Comput. Sci.*, Toronto, Canada (1986) 77–86.
[14] C.D. Thompson and H.T. Kung, Sorting on a mesh-connected parallel computer, *Comm. ACM* **20** (4) (1977).
[15] S.H. Unger, A computer oriented towards spatial interaction, *Proc. IRE* **46** (1958) 1744–1750.