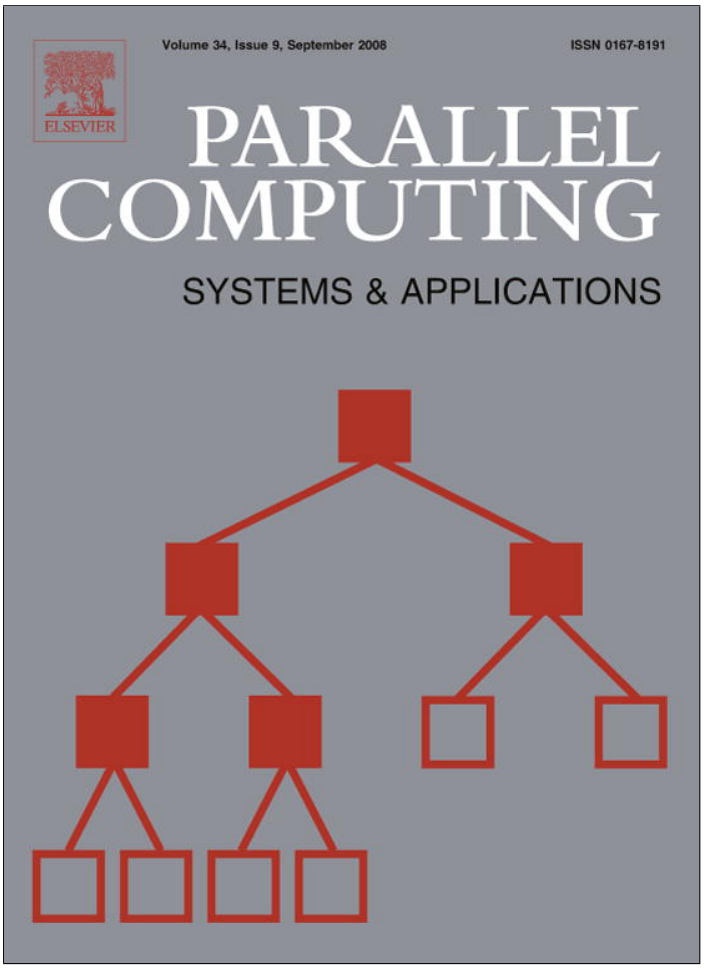

Parallel Computing 34 (2008) 487–496

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

A linear-time algorithm for computing collision-free path on reconfigurable mesh

Dajin Wang *

Department of Computer Science, Montclair State University, Upper Montclair, NJ 07043, USA

ARTICLE INFO

Article history:
Received 20 September 2005
Received in revised form 29 May 2007
Accepted 11 March 2008
Available online 18 March 2008

Keywords:
Image processing
Motion planning
Parallel algorithms
Path planning
Reconfigurable mesh
Robotics

ABSTRACT

The reconfigurable mesh (RMESH) is an array of mesh-connected processors equipped with a reconfigurable bus system, which can dynamically connect the processors in various patterns. A 2D reconfigurable mesh can be used to solve motion planning problems in robotics research, in which the 2D image of robot and obstacles are digitized and represented one pixel per processor. In this paper, we present an algorithm to compute a collision-free path between two points in an environment containing obstacles. The time complexity of the algorithm is $O(k)$ for each pair of source/destination points, with $O(\log^2 N)$ preprocessing time, where k is the number of obstacles in the working environment, and N is the size of the reconfigurable mesh.

1. Introduction

The general purpose of robot motion planning is to plan the movement of a robot in a known or unknown environment filled with obstacles. The planned movement is often subject to certain requirements. In a typical case, the robot will be commanded to navigate among a collection of static or time-dependent obstacles (i.e., moving obstacles). The shapes of those obstacles may range from very simple ones (such as a single segment) to arbitrarily complicated objects. The robot has to move from an initial position to a target position without colliding (but may be allowed to touch) with any of the obstacles. Moreover, there can be many different restrictions on the types of movement of the robot, and the robot's mechanical limitations have to be taken into account when planning its motion. For example, for a non-circular robot whose workspace is two-dimensional, its movement can be either translational or rotational. Thus one possible restriction is that only translational movements are allowed. Efficient algorithms were proposed for solving motion planning problems in past years [18–20].

A very important, constant problem a mobile robot needs to solve, and solve quickly, is path planning. Since path planning problems, especially in an environment with static or changing obstacles, are computationally intensive, parallel algorithms with various underlying architectures have been proposed. Tzionas et al. presented parallel algorithm for collision-free path planning for a diamond-shaped robot and its implementation in VLSI [19]. Jenq and Li have developed optimal algorithms for computing the *configuration space* (will be introduced in Section 3.2) by using hypercube computers [7]. Their algorithms run in $O(\log n)$ time for an $n \times n$ image by using $n \times n$ processors, and have been shown optimal for hypercube computers. Dehne et al. presented a systolic algorithm for computing the configuration space obstacles in a plane for a rectilinear convex robot [5]. The algorithm takes $O(n)$ time for an $n \times n$ image on an $n \times n$ mesh-connected computer. In the application of meshes for

* Tel.: +1 973 655 7615; fax: +1 973 655 4164.
E-mail address: wang@pegasus.montclair.edu

0167-8191/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2008.03.002

path planning, robot and obstacle images taken by the camera are represented in mesh processors, with each processor holding one pixel of the image. The image data can then be processed in parallel.

In this paper, we use the *reconfigurable mesh* to compute a collision-free path. We assume a two-dimensional working environment. The image of robot and obstacles are digitized, input and stored in the reconfigurable mesh, with one processor holding one pixel of the image. The obstacles we deal with are supposed to be disjoint convex or concave polygons (if two polygon images intersect, they are considered one polygon). The task is to navigate the robot from a source location to a target location, avoiding collision with any obstacles. The reconfigurable mesh architecture we choose to solve the aforementioned path planning problem is the one first proposed by Miller et al. [12,13]. It is an array of mesh-connected processors with a reconfigurable bus system that can reconfigure the whole mesh into different substructures. The major attractive feature of the reconfigurable mesh is its ability to communicate from one PE to any other PE in just one or two steps. That major feature makes it an attractive structure for a host of, especially geometry-natured, problems. The size of the problem instances often is not a hurdle for a fast solution.

With an $O(\log^2 N)$ preprocessing time for the given obstacle image, the proposed algorithm uses $O(k)$ time to compute a path for a pair of source/destination while avoiding all obstacles in the environment, where N is the total number of processors (pixels) and k is the number of disjoint obstacles.

The rest of this paper is organized as follows: In Section 2 we review the past works in the research of reconfigurable meshes. In Section 3, we describe in detail the structure of RMESH, the reconfigurable mesh computer we use to carry out our algorithm. In Section 3, we also briefly introduce the concept of configuration space, which is a technique used to expand (inflate) the obstacles in the working environment, so that the robot can be reduced to a point. Throughout this paper, we assume the obstacles are already expanded, so our robot is a point in plane. In Section 4, we introduce some operations on RMESH developed by other researchers. These operations will be made use of in our algorithm. Section 5 describes the algorithm to compute a collision-free path using reconfigurable mesh and analyzes its complexity. Concluding remarks are given in Section 6.

2. Previous related works

The algorithm proposed in this paper makes use of the *reconfigurable mesh* multiprocessor computer [12,13], which is an array of mesh-connected processors with a reconfigurable bus system that can reconfigure the whole mesh into different substructures. Since its first proposal more than a decade ago, the reconfigurable mesh model kept drawing researchers' interest. In [11], the list ranking problem is shown to be solvable on reconfigurable mesh using $O(n^{1+\epsilon})$ processors, where ϵ is an arbitrary positive constant <1. In [2], Bourgeois and Trahan presented simulation results for a host of reconfigurable optically pipelined bus models, and established that they possess the same complexity. So any of these models can simulate a step of one of the other models in constant time, regardless of the increase in instance size. Estrella-Balderrama et al. dealt with fault-tolerance on the reconfigurable meshes [6]. A novel technique was proposed to identify a healthy sub-mesh from a faulty model. In [16], an efficient Hough transform algorithm on a 3D reconfigurable mesh was proposed that runs in constant time on a 3D $n\log^2 N \times N \times N$ reconfigurable mesh. In [4], efficient algorithms were proposed for packet routing on the reconfigurable linear array and the reconfigurable two-dimensional mesh. The time bounds were shown to be better than those achievable on the conventional mesh and previously known algorithms. In [3], Chung and Chen presented an efficient parallel algorithm for solving the problem of neighbor-finding for a bincode-based binary image. The proposed neighbor-finding algorithm can run in $O(1)$ time on an $n^{1/2} \times n^{1/2} \times n^{1/2}$ reconfigurable mesh, using $O(n^{3/2})$ processors. The work of [15] included a comprehensive study of the important maze-routing problem, which can find applications in both VLSI routing and robotic path planning. RMESH architectures were shown to be particularly suitable for fast algorithms to solve maze-routing problems. To an extent, the current paper could be viewed as an extension of the work in [15] if the "mazes" are perceived as obstacles with special properties.

3. Preliminaries

3.1. Reconfigurable mesh

There are several different models of reconfigurable meshes [1,12,17,21]. The particular reconfigurable mesh architecture we consider in this paper is called RMESH [12,13]. It employs a *reconfigurable bus* to interconnect all processors. Fig. 1 shows the conceptual structure of a 4×4 RMESH, in which a square represents a *processor*, a circle represents a *switch*. The switches can be programmed to open/close in any specified manner. By doing so the interconnection bus may be reconfigured into smaller sub-buses that connect a subset of processors. The extra-equipped bus system is the most important feature of RMESH.

The major features of an RMESH are as follows:

- A 2D RMESH is a two-dimensional mesh-connected, $m \times n$ array of processors. We define $N = m \times n$ to be the *size* of the RMESH. The id of each processor is a 2-tuple (i,j), where i is its row index, j its column index. We designate that the id of the lower left corner processor be $(0,0)$.

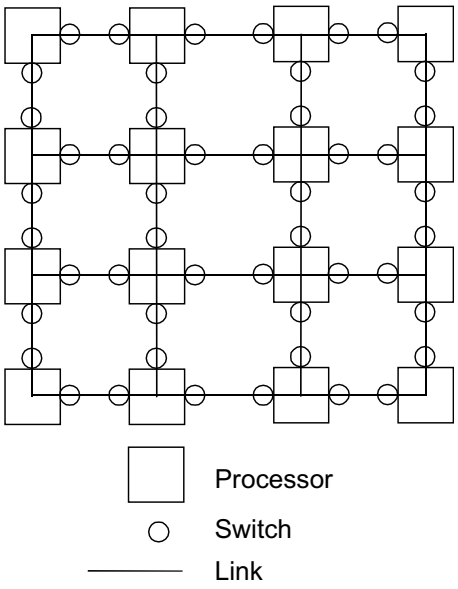

Fig. 1. A 4×4 RMESH. By setting bus switches, the processors can be reconfigured into different substructures.

- Processors are connected to each other with bus switches. Each processor has up to four switches (see Fig. 1), named E- (east), W- (west), S- (south), and N- (north) switches, respectively. These switches are software controlled and can be used to reconfigure the bus into sub-buses.For example, row buses are formed if each and every processor disconnects its S- and N-switches, and connects its E- and W-switches. Column buses are formed by disconnecting all E- and W-switches, and connecting all S- and N-switches. An all-plane bus can be formed by connecting all switches. Once a sub-bus is formed, all data move is among processors connected to this sub-bus.
- Only one processor can put data onto a given sub-bus at a time.
- In *unit* time, data put on a sub-bus can be read in parallel by every processor connected to the sub-bus, i.e., data are transferred using only switches without stopping at intermediate processors.

Mesh-connected parallel computers have been used to solve image processing problems [14,13], in which the images are digitized, input and stored in the mesh with one processor holding one image pixel. RMESH can be used to solve these problems much faster with its extra communication and reconfiguration power.

3.2. Obstacles in configuration space

In robotics research, an important problem is to compute a path from a source position to a target, avoiding all obstacles in the environment. To simplify the problem, we often tackle the problem with the assumption that the robot is a point in the plane. This assumption makes the observation to problems much clearer and easier, greatly reducing the complexity of design and analysis. To accommodate to real problems, in which robots are usually not point-like, we resort to the idea of *configuration space* [5,7,10]. To put it simply, a configuration space is obtained by expanding the original obstacles according to the robot's shape and its pattern of movement. After computing configuration space, a path planning problem with non-point-like robot and a set of obstacles can be reduced to a problem with a point-like robot and a set of expanded obstacles in configuration space.

In a simple example shown in Fig. 2, a rectangular robot R is to be moved to d. The two shaded polygons are the original obstacles. Suppose the right lower corner s is the "reference point" that we use to compute the path. Then the expanded obstacles in configuration space are the areas enclosed by the dark dashed lines and the original obstacles. After computing expanded obstacles, the path planning problem for robot R can be reduced to a problem for reference point s. In [8,9], the computation of configuration space obstacles have been considered. For robots of basic shapes (such as circles, polygons), the expanded obstacles can be obtained in $O(1)$ time. In this paper, we assume the obstacles are already expanded, so our robot is a point in plane.

4. Previous techniques

The collision-free path computation algorithm we propose in this paper will make use of some techniques introduced in [13]. These techniques and their time complexities are stated in this section. The detailed description and analysis can be found in [13].

Performing "OR" on a row/column of boolean values.

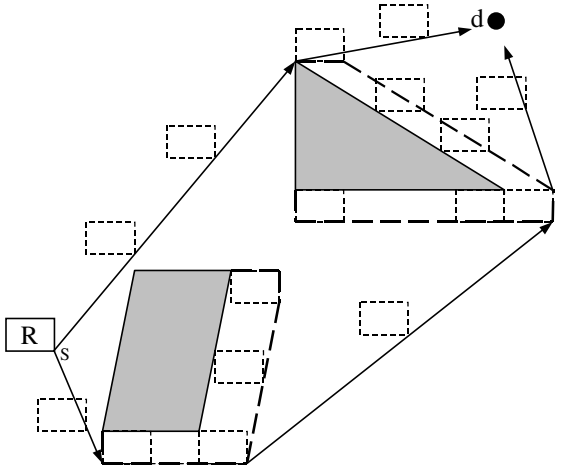

Fig. 2. R is a rectangular robot. The original obstacles are expanded so that R can be reduced to a point s. The working space after expansion is called configuration space.

A row/column of boolean values (1/0) can be ORed in $O(1)$ time on RMESH. Let processor $P_{i,j}$ at row i contain boolean value b_j, $1 \leqslant j \leqslant n$. After a fixed number of operations, the OR-result of all b_j can be collected at $P_{i,0}$ (and can be broadcast to any other processor if wanted).

Determining the maximum (or minimum) value of a row/column of data.

This useful operation can be done in $O(1)$ time. Let a row of elements $(x_1, x_2, \ldots, x_n)$ reside at the bottom row of a RMESH. Then after a fixed number of operations, the maximum (or minimum) value of $(x_1, x_2, \ldots, x_n)$ can be found – the processor $P_{0,j}$ containing the maximum value x_j will be aware of this fact. The method uses $n \times n$ processors and the column-OR operation stated above.

Determining the tangent lines from a point to a convex polygon.

See Fig. 3. Given a convex polygon G mapped on RMESH, and a point A outside of G. The two tangent lines from A to G can be determined in constant time on RMESH.

Enumerating the extreme points of the convex hull of a polygon.

See Fig. 4. Given a polygon G mapped on RMESH, we want to identify the extreme points of the convex hull of G. The enumerated extreme points can completely represent a polygon on plane. An enumerated extreme point stores its own location and number, and the locations and numbers of its preceding and following extreme points.

The enumeration can be done in $O(\log^2 N)$ time on an RMESH, where N is the size of the RMESH.

Finding the two tangent lines of two convex polygons.

See Fig. 5. Given two *disjoint* convex polygons G and H, mapped on RMESH. The two tangent lines of G and H can be determined in $O(1)$ time. The operation works out the convex hull of G and H.

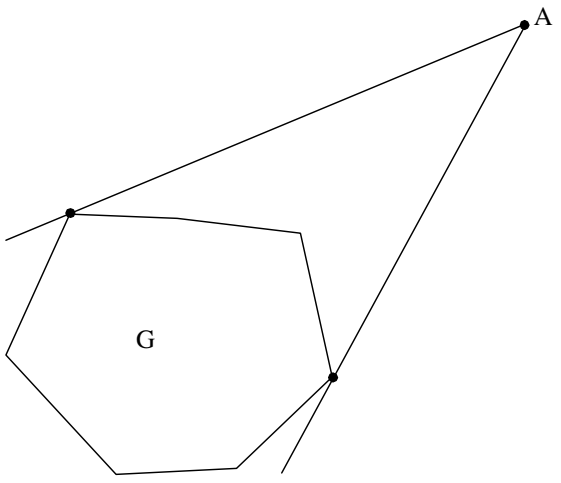

Fig. 3. The two tangent lines from A to G can be determined in $O(1)$ time.

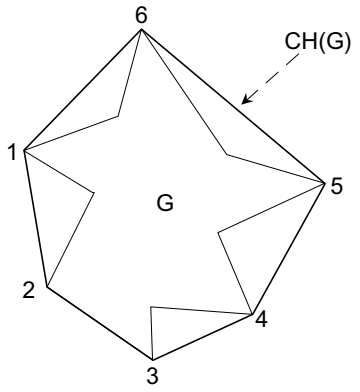

Fig. 4. The dark segments form G's convex hull. The convex hull extreme points of all polygons mapped on an RMESH of size N can be enumerated in $O(\log^2 N)$ time.

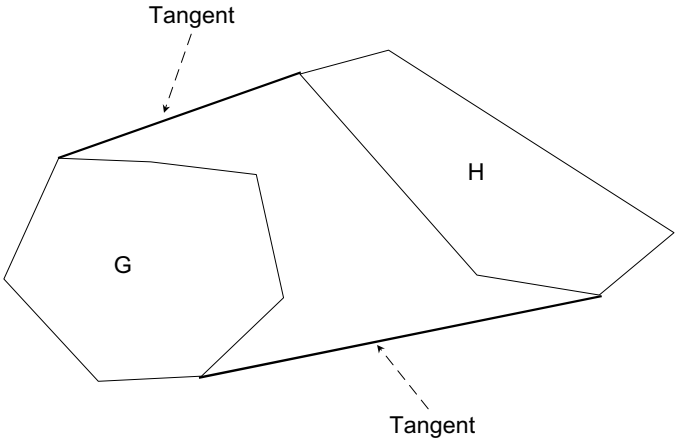

Fig. 5. G and H are two disjoint convex polygons. The two tangent lines of G and H can be determined in $O(1)$ time.

5. Collision-free path computation

5.1. Basic operations

5.1.1. Set handling

We designate a certain number of processors to represent a set, with each processor representing an element of the set. The unique location id can be used as the element id. A membership flag indicates whether the element is in the set or not (1 or 0). The following operations will be employed by our algorithm.

Determining whether the set contains at least two elements.

We first consider the case in which the element space contains only one row, say row i.

Step 1 Every "1" element broadcasts its id toward west. The id of the westernmost "1" element $P_{i,w}$ will be received by $P_{i,0}$; if no id is received, the answer is No (empty set); if an id is received, goto Step 2.

Step 2 $P_{i,0}$ temporarily sets $P_{i,w}$ to be a "0" element.

Step 3 Repeat the broadcasts in Step 1; if $P_{i,0}$ received another id, the answer is Yes, otherwise the answer is No (singleton set).

Step 4 $P_{i,0}$ sets $P_{i,w}$ back to "1".

Each step costs constant time, therefore the total cost is $O(1)$.

If the element space contains several adjacent rows $i, i+1, \ldots, j$, the procedure is described as follows:

Step 1 All rows run in parallel the 4-step single-row procedure; the results c_k ("0" for empty, "1" for singleton, "2" for at least 2) are stored at $P_{k,0}$, $i \leqslant k \leqslant j$.

Step 2 At column 0, all "1" and "2" processors broadcast their id and c_k toward south. The id and c_s of the southernmost "1" or "2" element $P_{s,0}$ will be received by $P_{i,0}$; if no id is received, the answer is No (empty set); if a $c_s = 2$ is received, the answer is Yes; if a $c_s = 1$ is received, goto Step 3.
Step 3 $P_{i,0}$ temporarily sets c_s at $P_{s,0}$ to be "0".
Step 4 All "1" and "2" elements broadcast toward south again; if $P_{i,0}$ received another id, the answer is Yes, otherwise the answer is No (singleton set).
Step 5 $P_{i,0}$ sets c_s at $P_{s,0}$ back to "1" or "2".

Again, all five steps can be accomplished in a unit step, hence $O(1)$ total cost.

Adding a new element.

We need to find a processor that has 0 flag, and allocate it to the new element. We assume the element space contains ith row and $P_{i,0}$ is the set handler.

Step 1 Every "0" element broadcasts its id toward west. The id of the westernmost "0" element $P_{i,w}$ will be received by $P_{i,0}$.
Step 2 $P_{i,0}$ sets $P_{i,w}$'s membership flag to "1".
Step 3 $P_{i,0}$ informs the processor that made the insertion request of the new element's id.

This operation obviously has complexity $O(1)$. The extension to multiple-row element space is straightforward and is omitted.

Deleting an element.

We need to set a specific processor's membership flag to "0". This is an easy operation. Again, we assume $P_{i,0}$ to be the set handler.

Step 1 $P_{i,0}$ sends a signal to target processor $P_{j,k}$.
Step 2 $P_{j,k}$ sets its membership flag to "0".

This operation obviously has complexity $O(1)$.

5.1.2. Merge (convex hull) of two non-disjoint convex polygons

In the proposed algorithm, we need to merge two convex polygons G, H into one. G and H are intersecting with a specific pattern, i.e., G intersects with exactly one known edge of H, as shown in Fig. 6. This merge can be done in constant time as follows:

Step 1 Using the method introduced in [13], a, b determine their tangent lines to G (dashed arrows labeled "1" in Fig. 6. Tangent points are c, d, e, f).
Step 2 Determine which two of c, d, e, f are *internal* to H. Method: compute which of the extended vectors $\vec{ad}, \vec{ae}, \vec{bc}, \vec{bf}$ intersect with a segment of H (the extended vectors $\vec{ae}$ and $\vec{bf}$ intersect with a segment of H, therefore e and f are internal).
Step 3 The *external* points c and d determine their tangent lines to H (dashed arrows labeled "2" in Fig. 6. Tangent points are g, h, i, j. Notice that "1" and "2" lines may coincide).
Step 4 The two segments that do not intersect are the final tangent lines ($\overline{dg}$ and $\overline{cj}$).

Every step can be done in fixed number of steps, hence $O(1)$ complexity. If G intersects with two known edges of H, then the above process can be repeated for the second edge. The complexity remains $O(1)$.

5.2. The algorithm

Before algorithm starts, we assume the digitized obstacle image has been stored in the RMESH, one pixel per processor. The images are black and white. A processor has a 1/0 flag indicating whether it is a black/white pixel. In the following algorithm description, $\overline{sd}$ represents the line segment from point s to point d; SG represents a set of polygonal obstacles; $CH(G_i, G_j)$ represents the convex hull of polygons G_i and G_j.

We illustrate the idea of the algorithm in Fig. 7. The algorithm first draws a straight line $\overline{sd}$ from source s to destination d. If $\overline{sd}$ intersects with any obstacles (polygons O_1 and O_2 in Fig. 7), the algorithm merges these obstacles by computing their convex hulls. The tangent lines (h_1 and h_2) used to form convex hulls may intersect with more obstacles (O_3 and O_4). The newly intersected obstacles are then merged again (h_3, h_4, h_5, and h_6). The above process is repeated, until the tangent lines are not intersecting with any obstacles, so that there is only one "big obstacle" G_f intersecting with $\overline{sd}$.

The algorithm then draws tangent lines from s to G_f and from d to G_f, respectively. The four tangent lines may intersect with more obstacles, which will be merged with G_f again. The merging process will be repeated until the four tangent lines do not intersect with any obstacles (t_1, t_2, t_3, and t_4 in Fig. 7). By now there are two feasible collision-free paths from s to d, and either one can be chosen as the algorithm output. The formal description of the algorithm follows.

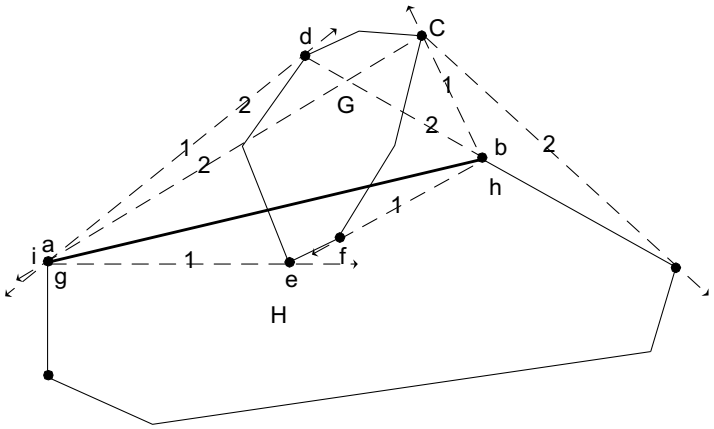

Fig. 6. Merge of two non-disjoint convex polygons.

5.2.1. Preprocessing

First, all polygonal obstacles are convexized and their extreme points enumerated. After the enumeration, every processor that is an extreme point contains the following information:

1. A flag that identifies itself as an extreme point.
2. The extreme point's circular number in the polygon.
3. The preceding/following extreme points' circular numbers and their locations (therefore the edge information is also available).
4. The id number of the polygon this point belongs to.

We assume that after convexization, s and d are not covered by any convex polygon. If that is not the case, a constant time treatment discussed later will reduce the problem to the assumed situation.

The enumeration takes $O(\log^2 N)$ time [13], where N is the size (number of processors) of the RMESH.

We then designate one or several "spare" rows (i.e., the processors that are not part of the obstacle images, for example, the rows at the bottom of the RMESH) to represent the set of all polygons. To facilitate $O(1)$ time buildup of this set, we can choose a processor directly under the first extreme point (i.e., the westernmost point) of the polygon as the set member. In case two extreme points have the same x-coordinate, the processor $P_{0,i}$ directly under them will represent the polygon of the lower point, $P_{0,i+1}$ will represent the polygon of the upper point. See Fig. 8 for illustration.

5.2.2. Collision-free path computation

We assume the working environment consists of $N = m \times n$ processors connected with RMESH structure. The obstacle images have been digitized, input and stored one pixel per processor. There are k disjoint polygonal obstacles.

Step 1 Start point s and destination point d broadcast their positions to all processors;
Step 2 Every edge calculates whether the segment $\overline{sd}$ intersects with itself;
Let $G_1, G_2, \ldots, G_m$ be polygons that intersect with $\overline{sd}$;
Step 3 Initialize $SG \leftarrow \{G_1, G_2, \ldots, G_m\}$;

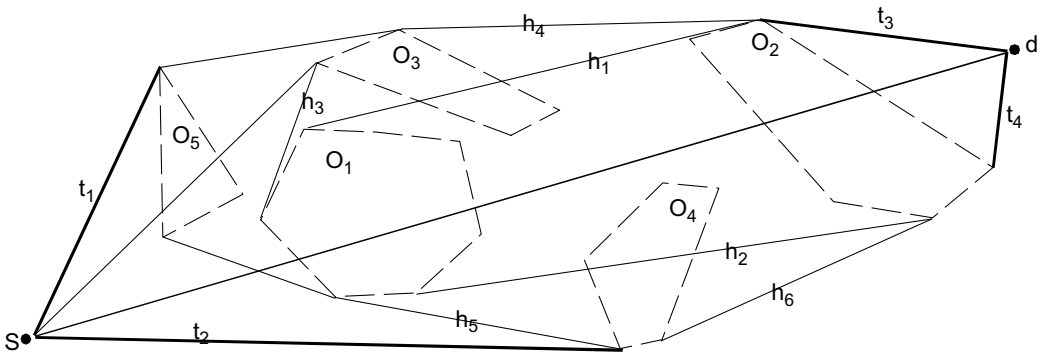

Fig. 7. Merges are repeatedly performed until the four tangent lines from s and d, respectively, do not intersect with any obstacles. As shown, there will be two feasible collision-free paths from s to d.

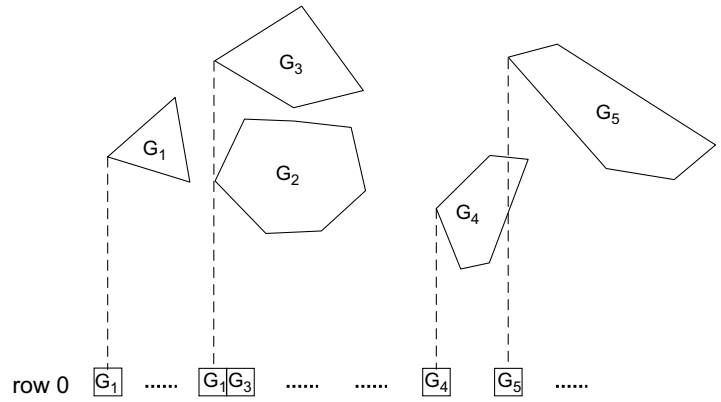

Fig. 8. A row of processors is used as the set to represent obstacles in the working space.

Step 4 **while** (*SG* contains 2 or more polygons)
$SG \leftarrow SG - \{G_i, G_j\}$; /* extract any two polygons G_i, G_j from *SG* */
Find the upper/lower tangent lines $t^u_{i,j}, t^l_{i,j}$ for G_i, G_j;
$G_k \leftarrow CH(G_i, G_j)$; /* G_i, G_j may or may not be disjoint */
$SG \leftarrow SG \cup \{G_k\}$;
$t^u_{i,j}, t^l_{i,j}$ broadcast their four end-points to all processors;
for every G_p such that $G_p \notin SG$ {
Calculate whether G_p intersects with $t^u_{i,j}, t^l_{i,j}$;
if Yes, **then** $SG \leftarrow SG \cup \{G_p\}$;
} end of for
} end of while
/* Entering Step 5, *SG* contains only one polygon */
Step 5 Let G_f be the polygon in *SG*;
Calculate the four tangent lines from s, d to G_f;
Broadcast end-points of the four tangent lines to all processors;
Step 6 **for** every G_p such that $G_p \notin SG$ {
Calculate whether G_p intersect with the 4 tangent lines;
if Yes, **then** $SG \leftarrow SG \cup \{G_p\}$;
} end of for
Step 7 **if** (*SG* contains two or more polygons) **then** goto Step 4 **else** goto Step 8.

/* Entering Step 8, the 4 tangent lines from s, d to G_f form a "big" convex polygon without intersecting with any more polygons in the environment */
Step 8 Starting from s, output the final polygon's extreme points in circular order, until d is output; the output points give a collision-free path; outputting in reverse circular order gives another path (see Fig. 7).

5.3. Time analysis

The preprocessing phase is clearly dominated by the convexization and extreme point enumeration of obstacles. So the complexity is $O(\log^2 N)$, where N is the size of the RMESH. For given environment, this computation is done only once. We then analyze the stepwise complexity.

Step 1: This step can be accomplished by two one-to-all broadcasts, therefore the complexity is $O(1)$.
Step 2: Every extreme point contains the segments information associated with it. Therefore the intersection status of all segments with $\overline{sd}$ can be computed in parallel, hence complexity $O(1)$.
Step 3: For every polygon, there are at most two segments intersecting with $\overline{sd}$; these two segments can send the "Yes" status to the polygon's representing point (the westernmost point) via the sub-bus *within* the polygon. The complexity is $O(1)$.The "Yes" polygons can vertically broadcast to a row representing set *SG*, setting membership flags of processors representing $G_1, G_2, \ldots, G_m$ to "1". The complexity is $O(1)$.
Step 4: The while-loop of this step will run at most k rounds, where k is the number of disjoint polygons in the environment. The condition can be checked in constant time using the method described in Section 5.1. We then look at the time complexity within each round.
$SG \leftarrow SG - \{G_i, G_j\}$: We can extract the two westernmost elements in *SG*. Constant time operation.

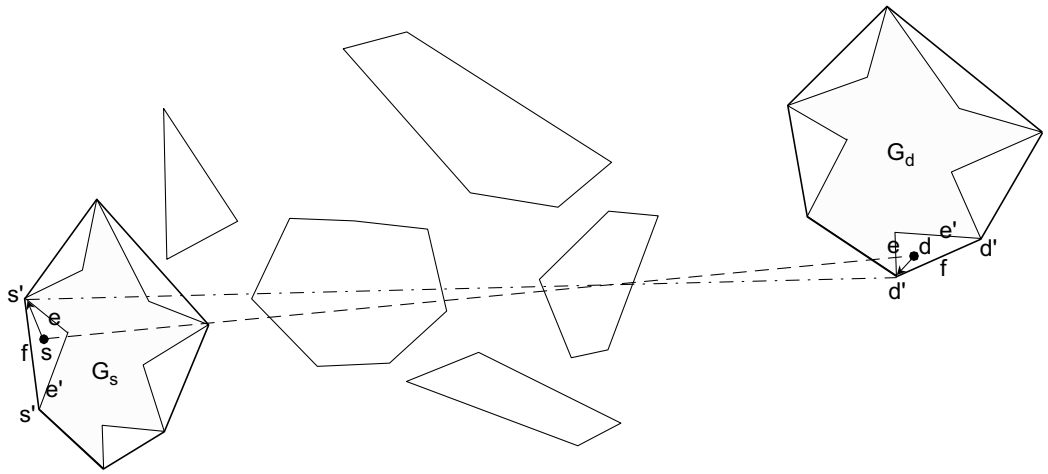

Fig. 9. After convexization, s and d are covered by the convex hulls of G_s and G_d, respectively.

Find the upper/lower tangent lines $t^u_{i,j}, t^l_{i,j}$ for G_i, G_j: Using the procedure introduced in [13], this can be done in constant time.
$G_k \leftarrow CH(G_i, G_j)$: G_i and G_j may or may not be intersecting with each other. Two disjoint polygons can be merged using the procedure introduced in [13]. Two intersecting polygons can be merged using the procedure described in Section 5.1. In both cases, merge can be done in constant time.
$SG \leftarrow SG \cup \{G_k\}$: Constant time operation.
$t^u_{i,j}, t^l_{i,j}$ broadcast their 4 end-points to all processors: Constant time operation.
for every G_l such that $G_l \notin SG$ {...}: All $G_l \notin SG$ can do the operations in parallel within the for-loop, therefore this for-loop is a constant time operation.
Summarizing Step 4, each round of while-loop takes $O(1)$ time to execute.

Step 5: Tangent lines calculation and broadcast both take constant time.
Step 6: Parallel for-loop with constant time operation, constant time in total.
Step 7: Condition checked in constant time using the method described in Section 5.1.
Step 8: All extreme points (segments) of the final polygon are part of two feasible collision-free paths. They can be broadcast in parallel to a row of processors.

In the worst case, the while-loop of Step 4 is run for every original obstacle, but each round of while-loop takes $O(1)$ time to execute. Therefore the algorithm has $O(k)$ complexity, where k is the number of obstacles in the working environment.

5.4. Concave obstacles near s and d

In the algorithm we just proposed, we assumed that after initial convexization, s and d are not covered by any convex polygon. In this subsection, we discuss the situation when this is not the case, such as shown in Fig. 9.

In Fig. 9, the original s and d are outside of concave polygons G_s and G_d. However, after preprocessing, they are covered by $CH(G_s)$ and $CH(G_d)$, respectively. The idea of solving this problem is also sketched in Fig. 9 – In preprocessing, we locate extreme points s' and d', and then call the algorithm for s' and d'. To locate s' and d', we need to keep the original pixels' "1" (black) flags of G_s and G_d. We also need to keep track of which original segments are eliminated, and by which new segment, during convexization (for example, in Fig. 9, when doing convexization, new segment f is obtained by eliminating e and e'). This information can be stored in processors holding eliminated extreme points. The operations are described below. They can be performed in fixed number of steps.

Step 1 In parallel, s and d broadcast their locations along four directions (E, W, N, and S);
Step 2 All "1" pixels who have received s and d send back their locations along the row or column buses; the location of pixel first received by s (d) is on a segment e of G_s (G_d);
Step 3 Find the segment f that eliminated e during enumeration; either of the two ends of f can be used as s' (d').

6. Conclusion

We have presented an algorithm on reconfigurable mesh (RMESH) to compute a collision-free path between two points in an environment filled with obstacles, in which the two-dimensional image of robot and obstacles are digitized and represented on RMESH one pixel per processor. The time complexity of the algorithm has been shown to be $O(k)$ for each pair

of source/destination points, with $O(\log^2 N)$ preprocessing time, where k is the number of obstacles in the working environment, and N is the size of the reconfigurable mesh (i.e., the number of pixels in the image). The most important feature of RMESH is its capability to transfer data among processors in constant time. In parallel computation of collision-free path, as in many other parallel algorithms, the majority of processing time is spent on moving data among processors. Making use of RMESH's constant time data transfer ability, our algorithms can be managed to run in $O(k)$ time.

We have used only two-dimensional RMESH for polygonal obstacles considered in this paper. For obstacles of more complicated shapes, one plane of processors my not be enough to compute collision-free path in the "linear" $O(k)$ time. If fast algorithms are still to be sought, we may need three-dimensional RMESH so that the massive data can be transferred on different planes in parallel.

There are many directions in which the work of this paper can be extended. An immediate one would be to develop efficient algorithms to find the *shortest path* between source and destination. Another future research can be on time-varying environments, in which obstacles move. The current algorithm deals with static obstacles. For moving obstacles, a space-time is needed to model the obstacles. The problem then can be turned into merging polyhedra in 3D space.

References

[1] Y. Ben-Ashen, D. Peleg, R. Ramaswami, A. Schuster, The power of reconfiguration, J. Parallel Distr. Comput. 13 (2) (1991) 139–153.
[2] A.G. Bourgeois, J.L. Trahan, Relating two-dimensional reconfigurable meshes with optically pipelined buses, in: IPDPS 2000, pp. 747–752.
[3] K.-L. Chung, H.-N. Chen, A neighbor-finding algorithm for bincode-based images on reconfigurable meshes, Comput. J. 43 (4) (2000) 315–324.
[4] J.C. Cogolludo, S. Rajasekaran, Permutation routing on reconfigurable meshes, Algorithmica 31 (1) (2001) 44–57.
[5] F. Dehne, A. Hassenklover, J. Sack, Computing the configuration space for a Robot on a mesh-of-processors, in: Proc. 1989 ICPP, vol. 3, 1989, pp. 40–47.
[6] A. Estrella-Balderrama, J. Fern ndez-Zepeda, A. Bourgeois, Fault tolerance and scalability of the reconfigurable mesh, in: Proc. IPDPS 2004, p. 172b.
[7] J. Jenq, W. Li, Computing the configuration space for a convex Robot on hypercube multiprocessors, in: Proc. 7th IEEE Symp. of Parallel and Distributed Processing, 1995, pp. 160–167.
[8] J. Jenq, D. Wang, Parallel computation of configuration space on reconfigurable mesh with faults, in: ICPP-2000 Workshop on High Performance Scientific and Engineering Computing, August 2000.
[9] J. Jenq, D. Wang, W. Li, Computing the configuration space on reconfiguration mesh multiprocessors, in: ISCA 13th Int. Conf. on Parallel and Distributed Computing Systems (PDCS-2000), August 2000.
[10] L. Kavraki, Computation of configuration-space obstacles using the fast Fourier transform, IEEE Trans. Robot. Automat. (1995) 408–413.
[11] S.-R. Kim, K. Park, Efficient list ranking algorithms on reconfigurable mesh, in: Proc. COCOON 2000, pp. 262–271.
[12] R. Miller, V.K. Prasanna Kumar, D.I. Reisis, Q.F. Stout, Meshes with reconfigurable buses, in: Proc. MIT Conf. Advanced Research in VLSI, April 1988, pp. 163–178.
[13] R. Miller, V.K. Prasanna Kumar, D. Reisis, Q.F. Stout, Parallel computations on reconfigurable meshes, IEEE Trans. Comput. 42 (6) (1993) 678–692.
[14] R. Miller, Q.F. Stout, Geometric algorithms for digitized pictures on a mesh-connected computer, IEEE Trans. Pattern Anal. Mach. Intell. PAMI-7 (1985) 216–228.
[15] H.-C. Lee, Efficient parallel algorithms on reconfigurable mesh architectures, Ph.D. Dissertation, University of Missouri-Rolla, 1996. <http://www.mis.yzu.edu.tw/faculty/hlee/csdiss/>.
[16] Y. Pan, Constant-time Hough transform on a 3D reconfigurable mesh using fewer processors, in: Proc. IPDPS 2000, pp. 966–973.
[17] M. Nigam and S. Sahni, Sorting n numbers on $n \times n$ reconfigurable meshes with buses, in: Proc. Int. Parallel Processing Symp., April 1993, pp. 174–181.
[18] K. Sutner, W. Maass, Motion planning among time dependent obstacles, Acta Inform. 26 (1988) 93–122.
[19] P. Tzionas, A. Thanailakis, P. Tsalides, Collision-free path planning for a diamond-shaped robot using two dimensional cellular automata, IEEE Trans. Robot. Automat. 13 (2) (1997) 237–250.
[20] D. Wang, Two algorithms for a reachability problem in one-dimensional space, IEEE Trans. Syst., Man, Cybern. 28 (4) (1998).
[21] B.F. Wang, G.H. Chen, F.C. Lin, Constant time sorting on a processor array with a reconfigurable bus systems, Inform. Process. Lett. (1990) 187–192.