Computing the Configuration Space on Reconfiguration Mesh Multiprocessors

John Jenq, Dajin Wang, and Wingning Li

***Abstract*— Configuration space computation is a transformation process that reduces a robot to a single reference point by expanding obstacles on the image plane. The obstacles can be expanded by inverting the robot along a reference point and then slide this reference point along their borders. The area covered by the union of inverted robot during the sliding along with the obstacles defines the configuration space of obstacles. This approach reduces a complex problem into a simple one. In this paper, we present a parallel algorithm for computing the configuration space obstacles by using reconfigurable mesh multiprocessors. The reconfigurable mesh multiprocessor system is a multiprocessor model with flexible bus connection capabilities. The digitized images of the obstacles and the robot are stored in an image plane. The algorithm takes O(1) time and is optimal.**

***Index Terms*— Configuration space, robotics, image processing, parallel algorithms, Reconfigurable mesh**

I. Introduction

CONFIGURATION space computation found applications in motion planning, computer graphics, robot-assisted surgery, automated assembly plans among many others. For example, Wytyczak-Partyka et. al[15]. propose no fly zone concept to assist surgeons. By defining the configuration space of the instrument, their system can provide a collision free working space for surgeons. In computer graphics application, Bandi and Thalmann adopted Configuration space approach to simulate human finger animation [1]. In [4], Ivanisevic and Lumelsky used configuration space as means to enhance human performance in teleoperation tasks. Because computing configuration space concept provides a generalized framework to study the motion planning problem and therefore is an important problem in path planning for automatic robotics applications see [3], [10], [11], [12], [13], [17].

Our aim in this paper is to develop constant time algorithm for computing the configuration space on reconfigurable mesh multiprocessors (RMESH). In [9], Kavraki used a Fast Fourier Transform based algorithm to compute configuration space obstacles. The objective of path planning is to find a path to move a robot A from a position s (the initial position) to another position d (the final position) without colliding with the obstacles already in space R. A common way to solve this problem is the configuration space approach which reduces the robot A to a single reference point p and expands each obstacle B_j to include all the positions of p that cause a collision between A and B_j. The expansion of an obstacle B_j is called the configuration space obstacle of B_j. In the new representation, the object A (robot) becomes a single point. The configuration space approach then effectively reduces a complex problem to a simple one.

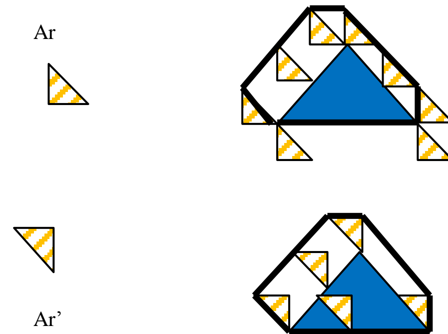

Fig. 1. Compute configuration space with robot inversion

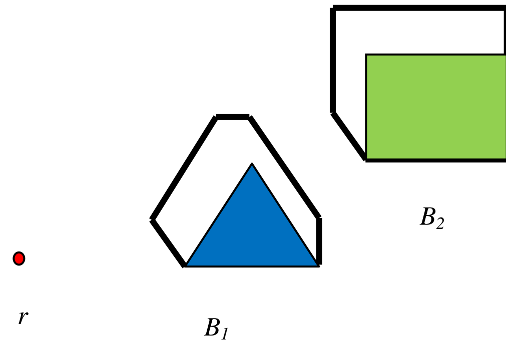

Fig. 2. A point robot r and the expanded obstacles B_1 & B_2

To calculate the configuration space obstacle of an obstacle B_j, one can firstly invert robot A, i.e. to rotate A about a reference point, say r, by 180° and then slide the reference point around the boundary of obstacle B_j. The union of the areas covered by A during the sliding, and the area originally covered by B_j defines the configuration space obstacle of B_j. Figure 1 shows a robot A with reference point Ar and the inverted robot with reference point Ar'. Figure 1 also shows the configuration space obstacle derived by using robot Ar and

Manuscript revised June 30, 2011.

John Jenq is with the Department of Computer Science, Montclair State University, Montclair NJ 07043 USA (phone: 973-655-7237; fax: 973-655-4164; e-mail: jenqj@ mail.montclair.edu).

Dajin Wang is with the Department of Computer Science, Montclair State University, Montclair NJ 07043 USA (e-mail: wangd@mail.montclair.edu).

Wingning Li is with the Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville AR 72701 USA (e-mail: wingning@uark.edu).

its inversion Ar' respectively. Figure 2 shows an example of two obstacles B_1 and B_2. The areas enclosed by the dark lines are the configuration space obstacles of B_1 and B_2. Note the triangular robot A becomes a point *r*.

Parallel algorithms targeted at different architectures had been proposed to speed up the whole process of path planning. For example, Dehne, Hassenklover, and Sack have presented a systolic algorithm for computing the configuration space obstacles in a plane for a rectilinear convex robot [2]. Their algorithm takes *O(N)* time for an $N \times N$ image on an $N \times N$ mesh computer. Tzionas, Thanailakis, and Tsalides have presented a parallel algorithm for collision free path planning of a diamond-shaped robot and its implementation in VLSI [16]. Jenq and Li developed optimal algorithms for computing the configuration space for circular, rectangular and convex robots by using hypercube computers [7], [8]. Their algorithms run in *O(logN*) time for an $N \times N$ image by using $N \times N$ processors and are optimal for hypercube computers.

In this paper, we consider convex robots and convex obstacles. The digitized bitmap image of a convex robot is a rectilinear convex polygon. Note the converse statement may not be true. A polygon is rectilinear convex if (1) the polygon is formed by horizontal and vertical line segments, and (2) the intersection of the polygon with any horizontal or vertical line consists of at most one line segment.

Since the class of reconfigurable mesh computers is a superset of the class of mesh computers, the algorithm developed by Dehne, Hassenklover, and Sack can be easily simulated with the same complexity, i.e., *O(N),* on a RMESH. In this paper, a constant time algorithm to compute configuration space on an $N \times N \times D$ RMESH is developed, where *D* is the diameter of the robot. We can achieve same time complexity and at the same time reduce the number of processor to $N \times N$ when the shape of the robot is either rectangular or circular.

We organize the remainder of the paper as follows. In section 2, we briefly describe the basic architecture and configuration of RMESH. In section 3, we list and develop some new fundamental RMESH data manipulation operations. These operations are functioned as building blocks on which the configuration space algorithms are developed. In section 4, the constant time algorithm for computing configuration space obstacles with a convex robot is discussed. We conclude this report in section 5.

II. Preliminaries on RMESH

The particular reconfigurable mesh architecture that we use in this paper is called RMESH[14]. It employs a reconfigurable bus to connect together all processors. Figure 3 shows a $4 \times 4 \times 2$ RMESH. By opening some of the switches, the bus may be reconfigured into smaller buses that connect only a subset of the processors. The flexible connection capability makes RMESH a powerful model to generate efficient solutions for various applications.

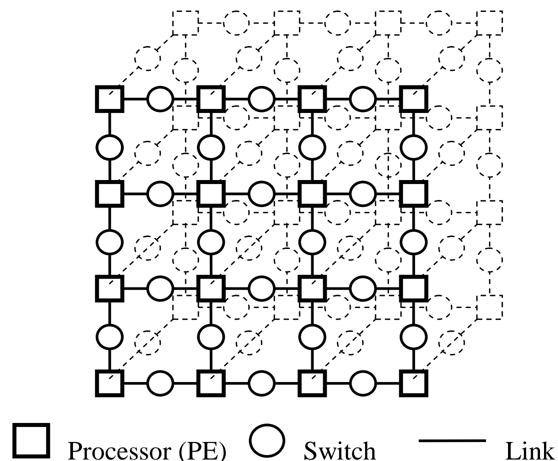

Fig. 3. A $4 \times 4 \times 2$ RMESH

The important features of an RMESH are:

1. An $N \times M \times L$ RMESH is a 3-dimensional mesh-connected array of processing elements (PEs). Each PE in the RMESH is connected to a broadcast bus, which is itself constructed as a $N \times M \times L$ grid. The PEs are connected to the bus at the intersection of the grids. Each PE manages up to six bus switches (see Fig. 3) that are software controlled and can be used to reconfigure the bus into sub buses. The ID of each PE is a triple (*i* , *j, k*) where *i* is the row index, *j* is the column index and *k* is the plane index. The ID of the upper left corner PE on plane zero is *(0,0,0)* and that of the lower right one is (*N-1,M-1,0*).
2. The six switches associated with each PE are labeled as E (east), W (west), S (south), N (north), B (back), and F (front). Notice that the east (west, north, south, back, front) switch of a PE is also the west (east, south, north, front, back) switch of the PE (if any) on its right (left, top, bottom, back, front). Two PEs can simultaneously set (connect, close) or unset (disconnect, open) a particular switch as long as the settings do not conflict. The broadcast bus can be subdivided into subbuses by opening (disconnecting) some of the switches.
3. Only one of the processors connected to a given subbus can broadcast its data on the subbus at any time.
4. In unit time, data put on a subbus can be read by every PE connected to it. Command broadcast(I) is used by a PE to broadcast the value in its register I to all the PEs on its subbus.
5. The statement R = content(bus) is used by a PE to read the content of the bus into its R register.
6. Row buses are formed when each processor disconnects (opens) its S switch, B switch, and connects (closes) its E switch. The column buses

can be formed by disconnecting the E and B switches, and connecting the S switch of each PE. Similarly, Z buses can be formed by connecting F (or B) switch and disconnecting E and S switches of each PE, while the plane buses can be formed when each PE only disconnects its B switch.

3.1. Broadcast

In a data broadcast operation, data originated in one PE are sent to the remaining *N -1* PEs, where *N* is the total number of PEs in the RMESH network. This operation takes $O(1)$ time.

3.2 Diagonalization

This operation will *diagonalize* a row (column) of elements, by which we mean moving a specific row (column) elements to diagonal positions with respect to that row (column). See Figure 4 for illustration. With the RMESH bus, this operation can be done in *O(1)* time.

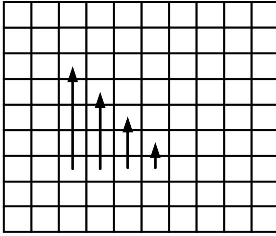

Fig. 4. Diagonalization of a row of 4 elements

3.3. Rank

Each PE(i) has a flag *selected*(i), which is set to true if PE(i) is selected. A rank operation assigns a rank to each PE, where the rank of PE(i), rank(i), is the number of selected PEs whose indices are less than i. This operation takes $O(logN)$ time. However, N elements on a single row can be ranked in *O(1)* time on an $N \times N$ RMESH [6].

3.4. Shift

Each PE has data in its *A* variable that is to be shifted to *B* variable of a processor that is *s* units, $s > 0$, to right or left in the same row (column). A variant of shift is the operation of *circular shift*, which performs shift with wrap-around. These operations can be done in $O(s)$ time. If $s = 1$ then the time becomes *O*(*1*). However, shifting a row of *m* elements for distance *s* can be done in *O*(*1*) time, if the $(m+s)\times s$ neighboring PEs are available to use. The procedure is given in Figure 5.

Step1 Partition the *m* elements into $\left\lceil \frac{m}{s} \right\rceil$ blocks.

Step2 Diagonalize each of the *s* elements upward onto the corresponding $s \times s$ block.

Step3 Form row subbuses for diagonal elements between even–odd blocks (i.e., block pairs ($0 \to 1$), ($2 \to 3$), …, etc.)

Step4 PEs on even blocks broadcast(*A*), where *A* is the value to be shifted.

Step5 PEs on diagonal of odd blocks do *B*=content(bus).

Step6 Column buses are formed on odd blocks.

Step7 PEs on diagonal of odd blocks broadcast(*B*)

Step8 The *s* elements on the bottom row of odd blocks do *B*=content(bus)

Step9 (Phase 2) Repeat Step4 through Step8 for odd-even blocks (i.e., block pairs ($1 \to 2$), ($3 \to 4$), …, etc).

Fig. 5. Constant time algorithm for shift operation.

Figure 6 shows the two-phase shift operation for $m = 20$ elements by using 24×4 PEs. The 20 PEs at bottom row are to be shifted 4 positions to the left. The 24×4 PEs are partitioned into six 4×4 blocks. The arrows represent data movement. If wrapped around shift is required then extra steps are needed to handle this. We omit the details here. The complexity can be easily seen to be *O(1)*.

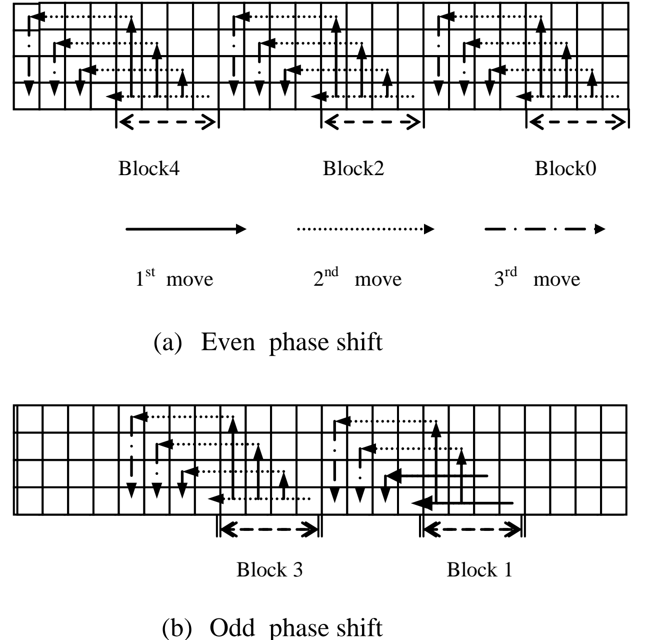

Fig. 6. Two phase shifting: (a) 1st phase (b)2nd phase

3.5. DrawSegment

This operation is defined only for PEs on the same row or column, for simplicity, we will use just one index to identify a processor, i.e., we use PE(i) to identify a processor in the implied row or column under consideration. Each PE(i) has a flag *mark*(i)*,* a variable *A(i)*, and another variable *ext*(i). PE(i) is marked if *mark*(i) true. A DrawSegment operation transmits the *A*(i) value of each marked PE(*i*), to PE(*i*), PE(*i+1*),...,PE(*i+ext(i)*) or stop propagating when the other PE whose *mark* value true is encountered. This implementation takes *O*(*1*) time as the following. Without loss generality, let us assume *A(i)* = *1* are the same for all marked PEs.

Furthermore, let us assume we will draw segments for the processors on the column 0 of plane 0 whose *mark*(i) is true and toward south. The procedure is in Figure 7

Step1 if *mark(i)* then disconnect N and B switches
Step2 if *mark(i)* then broadcast(i)
Step3 *index(k)* = content(bus), for $0 \le k < N$
Step4 if *mark(i)* then broadcast(*ext(i)*)
Step5 *ext(k)* = content(bus) for $0 \le k < N$
Step6 if $(index(k) + ext(k) \ge k)$ and $(mark(k) = false)$
then $\{mark(k) = true; A[k] = 1\}$, for $0 \le k < N$

Fig. 7. A constant time DrawSegment operation

Step1 form the column buses for the one dimensional RMESH under consideration. Step2 through Step5 send the row index of marked PE and its intended *ext*. value downward. Step6 is to determine which PEs are inside the range of the *ext*. of the marked PE above. For those PEs who are inside the range of the *ext*. set their *A* values.

3.6. AdjacentUnion

This operation is similar to the DrawSegment operation except that the *A*(i) is always of value *1*. The other difference is that when i+*ext*(i) of PE(i) is greater than j, *for mark(j) = true* and $j > i$, the *A[i]* value(which is one) continue propagating until PE with index *i+ext(i)* is encountered, while in DrawSegment operation the propagating value *A[i]* stops when PE(*j*) is encountered. This implementation takes $O(logN)$ time when there are N processors and can be done by recursive doubling on the size of the column buses and update the *ext*. values downward. It is similar to the hypercube operation used in [5] to compute the area of MAT. Since we are concerning constant time algorithms, there are two ways one can do to reduce the complexity to *O(1)*. Case (1) If the extended lengths *ext(i)* are the same for the participant PEs, and case (2) If there are at least $N \times ext(i)$ PEs available.

Let us examine these two methods separately. For case 1, DrawSegment can be used to perform the task. The rational is that during the drawing of the segment when a marked PE is encountered the propagation stops. Fortunately, the uncovered portion, that shall be drawn will be covered (drawn) by the encountered PE (which will draw the same value). This is exactly the dominate property mentioned in [7].

As for case 2, we assume there are $N \times \max(ext(i))$ PEs available; where the max*(ext(i))* is the diameter, *D*, of the robot. Let us assume that all the *N* PEs participating in the operation are in the same row. We firstly partition the *N* processors into $\lceil N/D \rceil$ partitions. The algorithm will run twice, one for the even blocks and the other for odd blocks. Each time when a block is processed, two blocks of processors are needed. Note each block is of size $D \times D$. The operation is very similar to the constant time shift operation mentioned earlier. The procedure is shown in Figure 8.

Step1 Diagonalization(*ext(i))*
Step2 DrawSegment(*ext(i)*) for PEs on the diagonal of the block
Step3 The PEs that are drawn from Step2 form column bus by disconnect N switch
Step4 Broadcast (1)
Step5 *A[i](0,j)*=content(bus)

Fig. 8. A constant time AdjacentUnion operation

Step1 transfers the *ext* values to the diagonal PEs by Diagonalization operation. Step2 draws segment for each PE on the diagonal line based on the *ext* value received from Step1. At Step4, the PEs that are marked by the DrawSegment operation broadcast the value one to the PEs at row 0. This can be done by firstly setting up the column bus as in Step3. The AdjacentUnion operation completes at Step5 when the *A[i]* is received, if there is any. Figure 9 shows an example for this operation. The numbers on the bottom represent *ext*. values. The horizontal arrow lines are DrawSegment operation, while the vertical arrow lines stands for broadcasting operation of Step4. Note in the example, after the AdjacentUnion operation, the PEs are all marked except the one that is in the rightmost position.

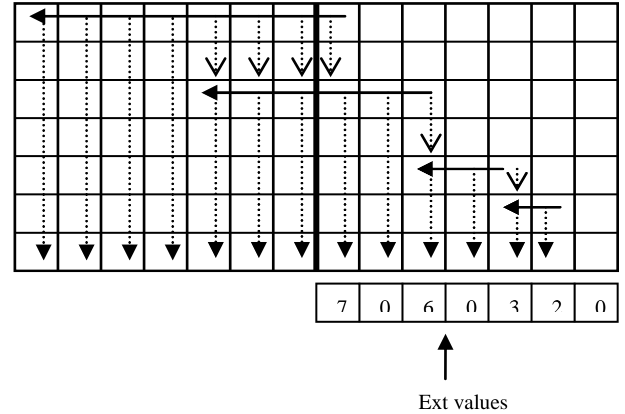

Fig. 9. AdjacentUnion operation on RMESH

3.7. Inversion

This operation rotates a rectilinear polygon by 180° around a given valid reference point (*i,j*). A reference point (*i,j*) is valid iff *i, j* are integers in the range of *0...N-1* and after the rotation the rectilinear polygon remains within the $N \times N$ image plane. This operation can be accomplished in constant time on an $N \times N$ RMESH provided the gray value of each pixel in the image of the rectilinear polygon is identical to one another. The procedure is outlined in Figure 10.

Step1 Reference point broadcasts its *i* and *j* indices to all pixels of the robot.
Step2 All right and left boundary pixels for the rectilinear polygon identify themselves.
Step3 Every right boundary pixel collects *length* information of its row segment and computes *newJ* index after the inversion.

Step4 The PEs corresponding to the right boundary pixels do diagonalization on window of $H \times H$ with the information of *newJ* and *length* information calculated at Step3, where H is the height of the rectilinear polygon.

Step5 Diagonal PEs broadcast *newJ* and *length*.

Step6 The PEs on the off-diagonal of the $H \times H$ window receive the information.

Step7 Setup row buses and broadcast *newJ* and *length*.

Step8 if received $newJ = j$ *index* for PEs in the $H \times H$ window then do DrawSegment(*length*).

Fig. 10. Inversion operation for a rectilinear polygon.

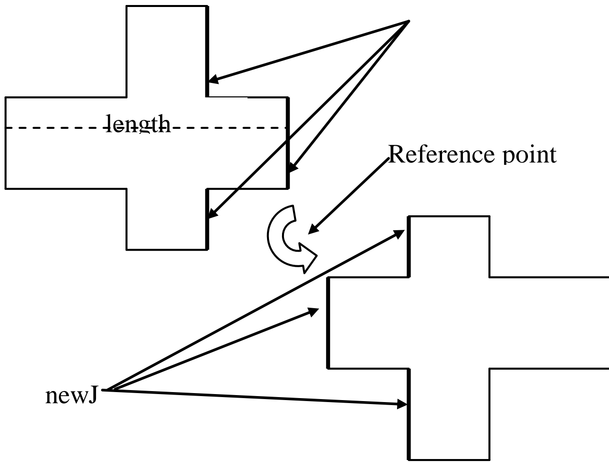

Fig. 11. Inversion of a rectilinear polygon

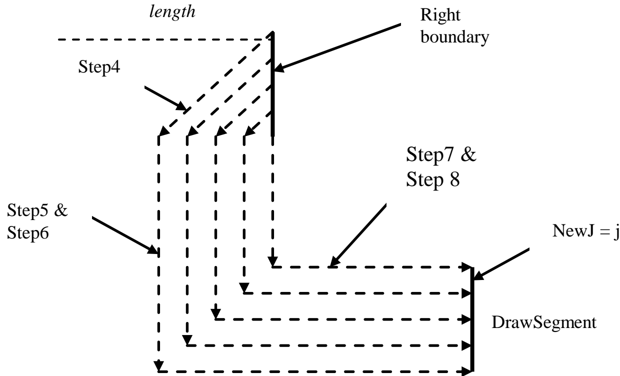

Fig. 12. Illustration of step3 to step8 of Inversion

Figure 11 shows the inversion process of a simple rectilinear polygon. Figure 12 illustrates Steps 3 through 8 of the procedure.

In Step1, the reference point PE, using the plane bus, broadcasts its coordinates (*i*,*j*) to all the PEs. Since the robot is rectilinear convex, it can be decomposed into a set of unit width horizontal segments. In Step2, after each PE checks its right and left neighbors, the PEs located at either ends of the horizontal segments can identify themselves. Once the left and right boundary PEs have identified themselves, they can determine the segment lengths for all the unit width horizontal segments. The length computation is accomplished by first setting up row buses, then each left boundary PE broadcasting the column index of its left neighbor on its row bus, and finally each right boundary PE receiving the index on its row bus and subtracting it from its column index to get the segment length. From laws of geometry, when a line segment is rotated by $180°$, the right end point of the line segment becomes the left end point of the rotated line segment. Since line segments are preserved under rotation, the length information of a line segment would be sufficient to reconstruct the rotated line segment if the coordinates of its left end point is known. The coordinates of the left end point is determined by the PE, currently located at the right end point of the corresponding not yet rotated segment, by applying the transformation matrix to its coordinates. This is done in Step3. After this step, the right boundary PEs need to send their segment lengths to the corresponding PEs located at the left boundary of the rotated segments. This can be accomplished by diagonalization operation, followed by column bus broadcast, and finally followed by row bus broadcast. These are done in Steps 4 to 7. Step8 reconstructs the polygon. The operations in each step can be done in parallel. The time complexity is *O(1).*

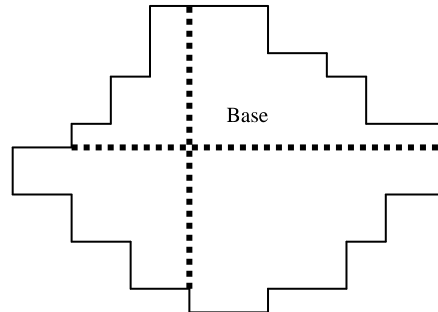

Fig. 13. A WBP convex robot partitioned into four L-shaped polygons

III. COMPUTATION OF CONFIGURATION SPACE ON RMESH

Using the fundamental operations developed in the previous section, we present a constant time RMESH algorithm to compute configuration space obstacles for those robots of which the digitized images may be modeled by WBPs (well behaved polygons). Briefly speaking, a WBP is a polygon that can be partitioned into at most four L-shaped polygons as shown in Figure 13. The reader is referred to [8] for a more detailed discussion of this type of polygons. Note that the digitized images of commonly encountered robot shapes, such as circles, rectangles, or convex polygons (possibly with rotation), are WBPs. The intersection of the two dotted lines, in Figure 13, is called the base point.

Some instances of the WBPs may have two base points. An example of such an instance is shown in Figure 14. A

technique of applying shift operations on the obstacles to reduce WBPs having two base points to that having one base point is developed in [8]. The same technique is used here. The reader is referred to [8] again for an elaborated discussion

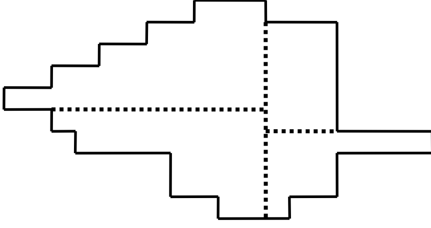

Fig. 14. A WBP convex robot with two base points

The computation of configuration space obstacles for a WBP shaped robot is reduced to that for a L-shaped robot. The final configuration space obstacles are computed by applying at most four iterations of the algorithm, that computes the configuration space obstacles for a L-shaped robot, and taking the union of the configuration space obstacles obtained from these iterations. Since the algorithms for the four different kinds of L-shaped robots are basically symmetrical, we only present the procedure for L-shaped robots having the base point at their upper right corner. Such a L-shaped robot is simply referred to as a robot in the remainder of the paper.

Before we proceed any further, let us note that the information describing the robot is needed by each obstacle PE, so that the PEs know how to expand the obstacles simultaneously, as if each obstacle has the robot slid around its boundary concurrently.

During the obstacle expansion, each obstacle first expands itself vertically, and then horizontally. For vertical expansion, each obstacle PE simply marks *H* PEs to its south as obstacle PEs, where *H* is the height of the robot. Once each obstacle PE receives the broadcasted *H* value, it can expands itself south-ward in *O(1)* time by applying the DrawSegment operation of Section 3.5.

Unlike vertical expansion, where before the expansion all obstacle PEs are the original obstacle PEs, horizontal expansion involves obstacle PEs that may be the original obstacle PEs or the new obstacle PEs due to vertical expansion. Hence, different horizontal expansion lengths may be required by different obstacle PEs.

For an original obstacle PE, the length of 0th (top) horizontal segment of the robot is used as its horizontal expansion length. For a new obstacle PE, its expansion length depends on its vertical distance from the original obstacle PE at the boundary. For a new obstacle PE, if this distance is *k*, the length of $k\text{-}1^{th}$ horizontal segment of the robot is used for its horizontal expansion. Thus, during horizontal expansion phase, each obstacle PE not only needs to know its vertical distance from the original obstacle PE (for an original obstacle PE this distance is 0), but also needs to know its horizontal expansion length.

In order to carry out the index (vertical distance) based retrieval of the length information in constant time, the following tiling procedure is developed. The procedure tiles the length and index of each horizontal segment of the robot for future reference, and operates on an $N \times N \times D$ RMESH, where *D* is the height of the robot, i.e., the number of unit width horizontal segments that the robot has. The tiling procedure is shown as in Figure 15.

Step1	Use shift operations to identify boundary pixel of the L-shaped robot
Step2	If right boundary pixel then setup row bus by disconnecting E, and B switches and broadcast its *j* index on the bus
Step3	*A*= content(bus) for PE(*i,0,0*)
Step4	If left boundary pixel then form row bus by disconnecting E, and B switches and broadcast its *j* index on the bus
Step5	*B*=content(bus) for PE(*i,0,0*)
Step6	For PE(*i,0,0*) that receive *A* and *B* do *runLength* := *A-B* and form Z bus
Step7	Rank from top to down for PEs that received *A* and *B*; put rank result in *R*
Step8	Broadcast(*runLength*) on Z bus for PE (*i,0,0*) from Step6
Step9	*runLength*=content(bus) for PE(*i,0,k*), where $0 \le k < s$
Step10	Broadcast(*R*) on Z bus for PE (*i,0,0*) from Step6
Step11	*D*=content(bus) for PE(*i,0,k*), where $0 \le k < s$
Step12	Form plane bus
Step13	Broadcast(*runLength*) for PE(*i,0,k*) *and k==D*
Step14	*runLength*=content(bus)

Fig. 15. Tiling of the length information for L-shaped polygon

Let $l_0, l_1, l_2, \ldots l_{d-1}$, be the lengths of the horizontal segments of a robot from top to bottom respectively. Let *rnuLength* be a register that each PE has. The goal of the tiling procedure is to assign l_i to all the *runLength* registers in plane i, $0 \le i \le D-1$. Of course, $l_0, l_1, l_2, \ldots l_{d-1}$, must be first calculated by the tiling procedure, and then distributed to different planes.

Step1 uses four shift operations to identify boundary PEs. If a PE is in the right most boundary of the robot(on its row) it sends its column index to the leftmost PE of the $N \times N$ RMESH on that row. This is done in Step2 and Step3. Similar the left most boundary PEs of the robot send their column indices to the leftmost PEs of the $N \times N$ RMESH on their rows. Step4 and Step5 fulfill this. Every leftmost PE then calculates the run length of the robot on that row. The next step is to rank the row strips of the robot starts from the top of the strip to the bottom(Step7). Note this operation is a special case of the general rank operation. Here the PEs involved in the ranking is in consecutive top to bottom fashion. Therefore the ranking operation can be done in *O(1)* time by first identify the top boundary PE(note the rank of this PE is 0). Followed by one broadcasting of the row index and simple algebra, other PEs can then determine their ranks. At this time the PEs that

are in the leftmost column of the $N \times N$ RMESH have the run length information of the robot on that particular row and the ranking information. These information will then broadcast to other planes by using Z bus. On receiving the run length information at Step9 and rank at Step11, the PEs can then compare the rank value with its *k* index. If these two values match then the PE will broadcast the *runlength* information to the PEs on its plane and this is done in Step12, 13 and 14 by using plane bus. It is easily seen the complexity is *O(1).*

Step0 Tiling;
Step1 compute *vertical-boundary(i,j,k)*;
    form column bus;
Step2 If *vertical-boundary(i,j,k)* then
        disconnect N switch;
        broadcast(*i*);
        temp(i,j,k) = content(bus);
    If not *obstacle(i,j,k)* then
        distance(i,j,k) =i-temp(i,j,k);
Step3 If *vertical-boundary(i,j,k)* then
    DrawSegment(*height(i,j,k)*) toward south;
Step4 If *A(i,j,k)* then *obstacle(i,j,k) = true*;
Step5 form z bus;
Step6 If *obstacle(i,j,k)* then broadcast(*distance(i,j,k)*)
        temp(i,j,k)= content(bus)
    if (*k*= = *temp(i,j,k)*) then broadcast(*runLength(i,j,k)*);
    runLength(i,j,k) = content(bus);
Step7 if *obstacle(i,j,k)* then
    AdjacentUnionRight(*runLength(i,j,k)*);

Fig. 16. Computing of configuration space for L-shaped robot

The algorithm to compute the configuration space obstacles is shown in Figure 16. The algorithm assumes that the digitized images of the obstacles and a robot are loaded into plane zero of the RMESH computer. During the image loading, two boolean variables, *robot* and *obstacle*, of each PE are initialized. A PE's *robot* variable is initialized to true iff it is a robot PE, i.e., it contains a pixel value of the robot. A PE's *obstacle* variable is initialized to true iff it is a obstacle PE. It is also assumed that the inversion operation has been performed and resulted in the robot under discussion.

Like all the algorithms presented in the paper, algorithm of Figure 16 is executed by every PE in the RMESH. Each PE(*i,j,k*) has the following important variables that are related to the current algorithm: *robot(i,j,k), obstacle(i,j,k), runLength(i,j,k), distance(i,j,k), height(i,j,k), vertical-boundary(i,j,k), A(i,j,k), and temp(i,j,k).* Each PE also has three constants *i,j,k*, which form the ID of the PE. Hence, in the algorithm symbols *i,j,k* refer to the constants *i,j,k* respectively.

The *temp* variable is used for obtaining bus data by each PE and does not have a significant role in the algorithm like the rest of the variables. The *obstacle* and *robot* variables are initialized during image loading as indicated earlier. In addition, the loading phase also initializes each *vertical-boundary* variable to false, *A* variable to zero, and *distance* variable to zero.

Variable *robot(i,j,k)* is only used in Step0 by the tiling procedure, which initializes the *runlength(i,j,k)* and *height(i,j,k)* variables of each PE. After Step0, the height of the robot is stored in the *height(i,j,k)* variable of each PE, and the length of the ith horizontal segment is stored in the *runLength(i,j,k)* variable of each PE in plane *i.* Thus, the values of *runLength(i,j,k)* variables of the PEs belonging to the same plane are the same.

In Step1, each obstacle PE checks its neighbor PEs to see if it needs to assign true to its *vertical-boundary(i,j,k)* variable. The shift operations are used for getting the values of the *obstacle(i,j,k)* variables of the neighboring PEs. Once this is done, the PEs set up the column buses for the next step. Step2 computes the values for the *distance(i,j,k)* variables. Each boundary PE sets up its column sub bus and sends its row index to the PEs down the south. Then each PE gets the row index from the bus and determines its distance to the boundary PE. The value of *distance(i,j,k)* will not be used later unless PE(*i,j,k*) is or becomes an obstacle PE. Step 3 carries out the vertical expansion. During the expansion, the value of *A(i,j,k)* will be set by DrawSegment operation if PE(*i,j,k*) is on the expansion path, i.e., PE(*i,j,k*) is a new obstacle PE. Step 4 reflects this fact by adjusting the *obstacle* variables. Step 5 prepares the Z buses so that the obstacle PEs can obtain their horizontal expansion length. Getting the length is done in Step 6. Step 7 carries out the horizontal expansion and completes the algorithm.

IV. CONCLUSIONS

Basic data manipulation operations on RMESH such as odd-even phase shifting operation, DrawSegment operation, AdjacentUnion operation, Image inversion operation were conceptualized and their implementation were developed. These operations may be used as basic building blocks to develop algorithms to solve more complex problems efficiently, which was demonstrated in this paper. Using these operations along with other existing operations a novel algorithm for computing configuration space obstacles was developed.

The algorithm we developed for computing the configuration space obstacles is for convex robot by using $N \times N \times D$ reconfigurable mesh with buses (RMESH), where D is the diameter of the robot under consideration. The algorithm is asymptotically optimal when time complexity is concerned. The algorithm runs in constant time and uses constant space. There are other interesting questions which we did not address in this report. Can we reduce the size of the RMESH and achieve the same optimal complexity? Can we compute configuration space obstacles when arbitrary shape robot is concerned? If yes, can we still achieve the constant time and space complexity?

REFERENCES

[1] S. Bandi, D. Thalmann, " A Configuration Space Approach for Efficient Animation of Human Figures", IEEE Computer Society Workshop on Motion of Non-Rigid and Articulated Objects, pp. 38 – 45, 1997.

[2] F. Dehne, A. Hassenklover, and J. Sack, "Computing the configuration space for a robot on a mesh-of-processors," Proceedings 1989 ICPP. vol. 3, pp. 40-47, 1989.

[3] Dinesh Manocha,Liangjun Zhang,Young J. Kim , "C-DIST: efficient distance computation for rigid and articulated models in configuration space", Proceedings of the 2007 ACM symposium on Solid and physical modeling, pp. 159-169, 2007.

[4] I. Ivanisevic, V. J. Lumelsky, "Configuration space as a means for augmenting human performance in teleoperation tasks", IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics", vol. 30, no. 3, pp 471 – 484, 2000.

[5] J. Jenq and S. Sahni, "Serial and Parallel Algorithms for the Medial Axis Transform", IEEE Transactions on Pattern Analysis and Machine Intelligence, Dec. pp 1218-1224, 1992.

[6] J. Jenq and S. Sahni, "Reconfigurable Mesh Algorithms for Fundamental Data Manipulation Operations" in Computing on Distributed memory Multiprocessors, NATO series F, ed. F. Ozguner, Spring Verlag, pp 27-46, 1993.

[7] J. Jenq and W. Li, "Optimal hypercube algorithms for robot configuration space computation", Proceedings of the 1995 ACM Symposium on Applied Computing, pp 182-186.

[8] J. Jenq and W. Li, "Computing the Configuration Space for a Convex Robot on Hypercube Multiprocessors", Proceedings of the 7th IEEE Symposium of Parallel and Distributed Processing, pp 160-167, 1995.

[9] L. Kavraki, "Computation of Configuration-Space Obstacles Using the Fast Fourier Trnasform", IEEE Transactions on Robotics and Automation, vol. 11(3), pp 408-413, 1995.

[10] C.L. Lia,1, K.W. Chanb, S.T. Tanb, "A configuration space approach to the automatic design of multiple-state mechanical devices", Computer-Aided Design 31, pp 621–653, 1999, Elsevier.

[11] T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," CACM, pp. 5609.-570, 197.

[12] T. Lozano-Perez, "Spatial planning: A configuration space approach," IEEE Trans. on Computers, pp. 108-120, 1983..

[13] T. C. Manjunath, Gopala, Ashok Kusagur, B. G. Nagaraja , "Simulation & Implementation of Shortest Path Algorithm with a Mobile Robot Using Configuration Space Approach", International Conference on Advanced Computer Control, pp. 197-201, 2009.

[14] R. Miller, V. Prasanna-Kumar, D. Reisis, and Q. Stout, "Parallel Computations on Reconfigurable Meshes", IEEE Transactions on Computers, vol. 42(6), pp 678-692, 1993.

[15] Andrzej Wytyczak-Partyka, Jerzy W. Rozenblit, Chuan Feng, Allan J. Hamilton, "Defining Spatial Regions in Computer-Assisted Laparoscopic Surgical Training", IEEE International Conference on the Engineering of Computer-Based Systems, pp. 176-183, 2009.

[16] P. Tzionas, A. Thanailakis, and P. Tsalides, "Collision-Free Path Planning for a Diamond-Shaped Robot Using Two dimensional Cellular Automata", IEEE Transactions on Robotics and Automation, vol.13(2), pp 237-250, 1997.

[17] Wang Yuquan, Zhu Qidan, Zhou Fang, Wang Tong, "Path Planning for Multi-Joint Manipulator Based on the Decomposition of Configuration Space", International Conference on Intelligent Computation Technology and Automation, pp. 661-664, 2009.

John Jenq is an associate professor of Computer Science Department at Montclair State University, Montclair New Jersey. Dr. Jenq received his Master of Science and PhD from University of Minnesota, Minneapolis in 1986 and 1991 respectively. His research interests include parallel and distributed computation, image processing, pattern recognition, data mining, algorithmic robotics, and internet applications . Dr. Jenq is a member of both ACM and IEEE.

Dajin Wang is a professor of Computer Science Department at Montclair State University. Dr. Wang received his B. Eng degree from Shanghai University of Science and Technology in 1982, Master of Science and PhD from Stevens Institute of Technology in 1986 and 1990 respectively. His research interests include interconnection networks, fault tolerant computing, parallel and distributed computing, wireless mobile and sensor networks, and algorithmic robotics.

Wingning Li is a professor with Department of Computer Science and Computer Engineering, University of Arkansas, where he has been serving from 1998-present. Dr. Li obtained his B.S. Degree in Computer Science University of Iowa, December 1982, his M.S. Degree in Computer Science University of Minnesota, November 1985, and his Ph.D. Degree in Computer Science University of Minnesata, September 1989. Dr. Li research interests are in the areas of Computer-aided design for VLSI circuits, combinatorial optimization, design and analysis of algorithms in both theoretical and experimental settings, parallel computing, software reuse and construction, and GUI design and development. Dr. Li is member of both ACM and IEEE.