Reconfiguration of Cube-Style Modular Robots Using $O(\log n)$ Parallel Moves

Greg Aloupis[1], Sébastien Collette[1⋆], Erik D. Demaine[2⋆⋆], Stefan Langerman[1⋆⋆⋆], Vera Sacristán[3†], and Stefanie Wuhrer[4]

[1] Université Libre de Bruxelles, {greg.aloupis,secollet,slanger}@ulb.ac.be
[2] Massachusetts Institute of Technology, edemaine@mit.edu
[3] Universitat Politècnica de Catalunya, vera.sacristan@upc.edu
[4] Carleton University, stefanie.wuhrer@gmail.com

Abstract. We consider a model of reconfigurable robot, introduced and prototyped by the robotics community. The robot consists of independently manipulable unit-square atoms that can extend/contract arms on each side and attach/detach from neighbors. The optimal worst-case number of sequential moves required to transform one connected configuration to another was shown to be $\Theta(n)$ at ISAAC 2007. However, in principle, atoms can all move simultaneously. We develop a parallel algorithm for reconfiguration that runs in only $O(\log n)$ parallel steps, although the total number of operations increases slightly to $\Theta(n \log n)$. The result is the first (theoretically) almost-instantaneous universally reconfigurable robot built from simple units.

1 Introduction

In this paper, we consider homogeneous self-reconfiguring modular robots composed of unit-cube *atoms* arranged in a grid configuration. Each atom is equipped with mechanisms allowing it to extend each face out one unit and later retract it back. Furthermore, the faces can attach/detach to faces of adjacent atoms; at all times, the atoms should form a connected mass. When groups of atoms perform the four basic atom operations (expand, contract, attach, detach) in a coordinated way, the atoms move relative to one another, resulting in a reconfiguration of the robot. Fig. 1 shows an example of such a reconfiguration. Each atom is depicted as a square, with a T-shaped arm on each side.

The robotics community has implemented this model in two prototype systems: *crystalline atoms* [3–5] and *telecube atoms* [6, 7]. In the crystalline model, the default state for atoms is expanded, while in the telecube model, the default state is contracted. Thus Fig. 1 reconfigures a crystalline robot, or an expanded

⋆ Chargé de Recherches du FRS-FNRS.
⋆⋆ Partially supported by NSF CAREER award CCF-0347776, DOE grant DE-FG02-04ER25647, and AFOSR grant FA9550-07-1-0538.
⋆⋆⋆ Chercheur Qualifié du FRS-FNRS.
† Partially supported by projects MEC MTM2006-01267 and DURSI 2005SGR00692.

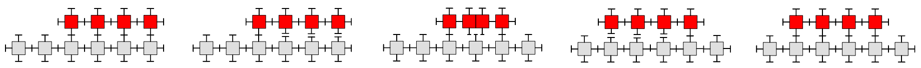
Fig. 1. *Example of reconfiguring crystalline atoms: the top row of atoms is able to shift to the left, using the bottom row of atoms as a fixed base.*

telecube robot. The crystalline robots work in a single plane, forbidding expand/contract/attach/detach operations parallel to the z axis, which is the case we consider in this paper.

To ensure connectedness of the configuration space, the atoms must be arranged in *meta-modules* (or simply *modules*), which are groups of $k \times k$ atoms. Any value $k \geq 2$ suffices for universal reconfigurability [2, 7]. Here the collection of atoms composing a robot must remain *connected* in the sense that its module graph (where vertices correspond to modules and edges correspond to attached modules) is connected.

The complexity of a reconfiguration algorithm can be measured by the number of *parallel steps* performed ("makespan"), as well as the total number of atom operations ("work"). In a parallel step, many atoms may perform moves simultaneously. The number of parallel steps is typically the most significant factor in overall reconfiguration time, because the mechanical actions (expansion, contraction, attachment, detachment) are the slowest part of the system.

Our main contribution in this paper is a reconfiguration algorithm that, given a source robot S and a target robot T, each composed of n atoms arranged in $k \times k$ modules for some constant k, reconfigures S into T in $O(\log n)$ parallel steps and a total of $O(n \log n)$ atom operations. This result improves upon the reconfiguration time of the algorithm presented at ISAAC 2007 [2], which takes $O(n)$ parallel steps (although only $O(n)$ total operations, and also for three-dimensional robots), as well as previous $O(n^2)$ algorithms [5, 7, 4].

A central assumption in our algorithm is that one atom, by contracting or expanding, can pull or push all n atoms (*linear strength*). Thus our algorithm certainly tests the structural limits of a modular robot, but on the other hand this assumption enables us to achieve reconfiguration times that are likely asymptotically optimal. The quadratic reconfiguration algorithms of [5, 7, 4] may be given credit for being the least physically demanding on the structure of the robot. Even the algorithm in [2] is less demanding than what we propose here, because it does not produce arbitrarily high velocities (although it still uses linear strength). Another recent algorithm [1] considers the case where atoms have only constant strength, and attains $O(n)$ parallel steps and $O(n^2)$ total operations, which is optimal in this setting. Thus the improvement in reconfiguration time obtained here *requires* a more relaxed physical model.

The main idea of our parallel algorithm is to reconfigure the given robot into a canonical form, by recursively dividing the plane into a hierarchy of square *cells* and employing a divide-and-conquer technique to merge quadruples of cells. Each merge creates a cell containing a simple structure using a constant number of moves. This structure, which fills the perimeter of a cell as much as possible, can be decomposed into a constant number of rectangular components. Because the steps to merge cells of the same level can be executed in parallel, the total number of parallel steps used to reconfigure any configuration to a simple structure is $O(\log n)$. The entire reconfiguration takes place in the smallest $2^h \times 2^h$ square containing the initial configuration, where h is an integer.

We choose to describe our algorithm in terms of the naturally expanded modules of crystalline robots. Of course, this immediately implies reconfigurability in the naturally contracted telecube model, by adding one step at the beginning and end in which all atoms expand and contract in parallel. We also expect that the individual constructions in our algorithm can be modified to directly work in the (2D) telecube model as well.

Our algorithm effectively uses modules of 4×4 atoms, but for clarity and brevity assumes that atoms initially appear in *blocks* of 32×32. Reducing the module size leads to more complicated basic operations that we have designed for use on large rectangular components. On the other hand, reducing the initial block size leads to a larger number of possible shapes that we must consider during the merge of cells. We have designed (though not rigorously analyzed) a range of algorithms for 2×2 modules with decreasing restrictions on block size. This is discussed in Section 5. However, the bulk of this paper focuses on the version that is easiest to describe.

2 Definitions

We will mainly deal with modules, not atoms, which can be viewed as lying on their own square lattice somewhat coarser than the atom lattice. Refer to Fig. 2 for examples of the following notions. In all figures, modules are depicted as squares unless mentioned otherwise. A module is a *node* if it has exactly one neighbor (a leaf node), more than two neighbors (a branching node), or exactly two neighbors not collinear with the node (a bending node). A *branch* is a straight path of (non-node) modules between two nodes (including the nodes themselves). A *cell* is a square of module positions (aligned with the module lattice), some of which may be occupied by modules. The *boundary* of a cell consists of all module positions touching the cell's border. For cells of sufficient size the *near-boundary* consists of all module positions adjacent to the cell's boundary. If a branch lies entirely in the boundary of a cell, we call it a *side-branch*. The configuration within a cell is a *ring* if the entire cell's boundary is occupied by modules, and all remaining modules within the cell are arranged at the bottom of the cell, filling row by row from left to right. The configuration within a cell is *sparse* if it contains only side-branches. A *backbone* is a set of branches forming a path that connects two opposite edges of a cell.

3 Elementary Moves That Use $O(1)$ Parallel Steps

Throughout this paper, whenever we describe a move, it is implied that we do not disconnect the robot and that no collisions occur. We first describe three

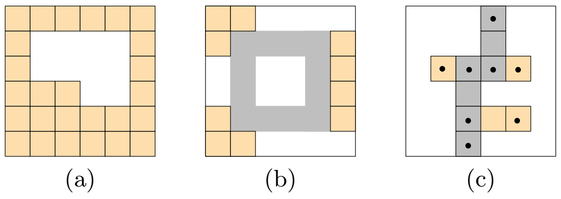

Fig. 2. *Definitions; modules are depicted by squares. (a) A ring. (b) A sparse cell with five side-branches and shaded near-boundary. (c) A shaded backbone and eight nodes.*

basic module moves (*slide, compress, k-tunnel*) that are used in [2]. We omit a detailed description of how to implement these moves in terms of individual atom operations. A *compression* pushes one module m_1 into the space of an adjacent module m_2. The atoms of m_1 literally fill the spaces between those of m_2 (see Fig 3). Any part of the robot attached to m_1 will be displaced by one unit along the same direction. Two modules can occupy the same position in the module lattice. A *decompression* can be applied to such a position, as long as an adjacent position contains enough space.

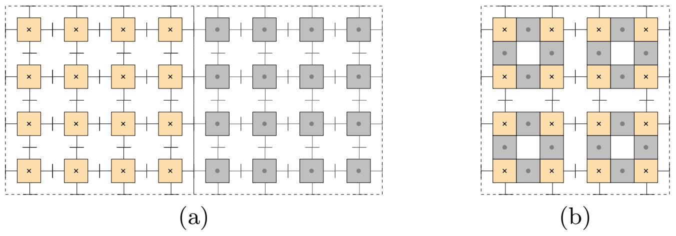

Fig. 3. *Compression of two adjacent 4×4 modules into one lattice position.*

A *slide* moves a module to an adjacent position , using two substrate modules. See Fig. 4a. The *k-tunnel move* compresses a leaf module into the robot, and *decompresses* another module out into a leaf position. An entire path of modules between the two leaves is involved in this move. Within each branch in this path, modules shift in the direction of the compression, and essentially transfer the compression to the next bend. Any modules attached to the branches will also shift. This issue is addressed later on. See Fig. 4b; The parameter k denotes the number of branches (or bends) in the path between the two modules. The move takes $O(k)$ parallel steps, but in our uses k will always be a small constant.

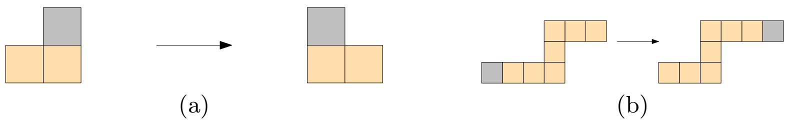

Fig. 4. *(a) Slide move; (b) Tunnel move.*

We now proceed to describe new basic moves that form the basis of our reconfiguration algorithm.

3.1 Staircase Move

The *staircase move* transforms a rectangle of $k_1 \times k_2$ modules to one of dimensions $k_2 \times k_1$, both sharing the same lower-left corner C. Connectivity to the rest of the robot is maintained through the module at C, and thus that module cannot move. Without loss of generality, we can assume that $k_1 \geq k_2$; otherwise, we invert the sequence of operations described.

First, we move every row of modules to the right using a slide move with respect to the row immediately below, as in Fig. 5(b). Second, we move every column that does not touch the top or bottom border of the bounding box down using a slide move, as in Fig. 5(c). Finally, we move every row to the left using a slide move, as in Fig. 5(d). Note that the sliding motions of each step are executed in parallel. Also, each operation can be done at the atom-level, as was shown in Fig. 1. Thus the move works even if $k_2 = 1$.

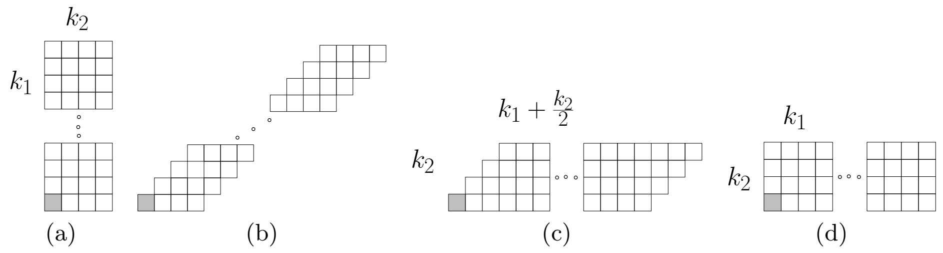

Fig. 5. *Staircase move in three parallel steps. The shaded module maintains connectivity to the rest of the robot.*

If we require that the transformation between rectangles takes place within the bounding box of the source and target configurations, we can modify the above procedure without much difficulty. This modification is omitted in the present version of this paper.

3.2 Elevator Move

The *elevator move* transports a rectangle of modules by k units between two vertical strips. Fig. 6(a) shows the initial configuration in which a rectangle is to be transported vertically downward. First we detach the top half T of the rectangle from the bottom half B. Furthermore, B detaches from the vertical strip on the right. Let R be the rightmost vertical column of k atoms along the left strip, together with the atoms to the left of B. We detach R to its left, except at the very bottom, and detach R above, thus creating a corner with B. Then we contract R vertically, thereby pulling B downward half way. This is shown in Fig. 6(b), in which, however, we have let R be a vertical column of modules instead of atoms, due to the large width of the shape. Thus far, T has maintained the connectivity of the robot. Afterward, B attaches to the right vertical strip and detaches from R, which is now free to expand and re-attach to the top, as in Fig. 6(c). Now R detaches from the bottom and contracts upwards.

It re-connects to B at the bottom, as in Fig. 6(d). In the last step, shown in Fig. 6(e), B detaches from the right side, and R expands, thereby moving B all the way to the bottom. At this point, B has reached its target position. It now assumes the role of maintaining connectivity, and the process can be repeated for T.

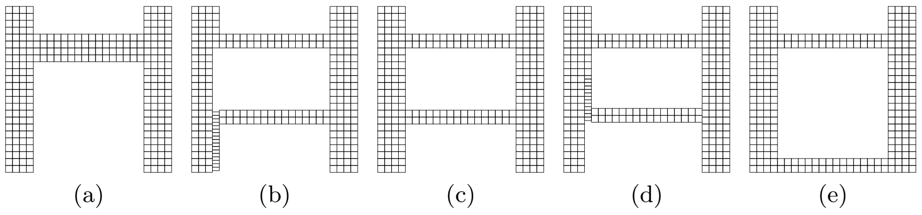

Fig. 6. *Elevator move in $O(1)$ parallel steps.*

3.3 Corner Pop

Consider a rectangle R of $k_1 \times k_2$ module units, where without loss of generality $k_1 \leq k_2$. Let R be empty except for a strip V of modules on its left border and a strip H along the bottom. The strips form a corner, as shown in Fig. 7(a).

The *corner pop* moves the modules in R to the upper and right borders of R. During this corner pop, the modules at the top-left and bottom-right corners of R do not move. It is assumed that only these positions connect to modules outside R. Thus, this operation preserves the connectivity of the robot.

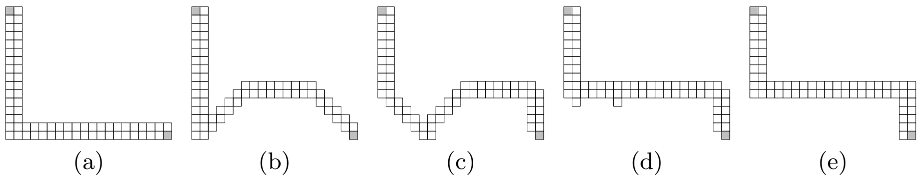

Fig. 7. *Popping a corner in $O(1)$ parallel steps. The shaded modules maintain connectivity to the rest of the robot.*

We first create two staircases of height $k_1/2$ at the two ends of H, as in Fig. 7(b). This shifts the middle of H upward. Next, we use the lower half of V to create a staircase of width $k_1/2$. Simultaneously, the rightmost staircase of H also moves so that it ends up on the right border of B, as in Fig. 7(c). We move the two remaining staircases upward, as in Fig. 7(d). Some simple cleaning up transforms this configuration into a symmetric canonical shape; see Fig. 7(e).

3.4 Parallel Tunnel Move

The *parallel tunnel move* takes as input a horizontal row H of modules together with, on the row immediately above, several smaller horizontal components that have no other connections. The top components are absorbed into H, after which H extends horizontally. Alternatively, the absorbed mass can be pushed out anywhere else on top of H, provided the target space is free. This move allows us to merge an arbitrary number of strips in the top row in $O(1)$ time.

The idea is to take all odd lattice positions along H and perform 1-tunnel moves, i.e., absorb modules from above and compress them under even positions. Then decompressing them all in parallel just expands H horizontally. Any modules remaining on top will shift over during the expansion, since they are attached to H. A gap will remain to the right of each such module, so we can repeat one more time to complete the move.

Fig. 8 illustrates half of the absorption of one module into H. Note that groups of 4 atoms move separately (they can be considered to be temporary smaller modules). As described, this procedure assumes that the bottom row is critically connected to other parts of the robot at one position, and absorbed modules are redirected away from that position. For 4×4 modules, this assumption is not required, but the minor implementation differences are omitted.

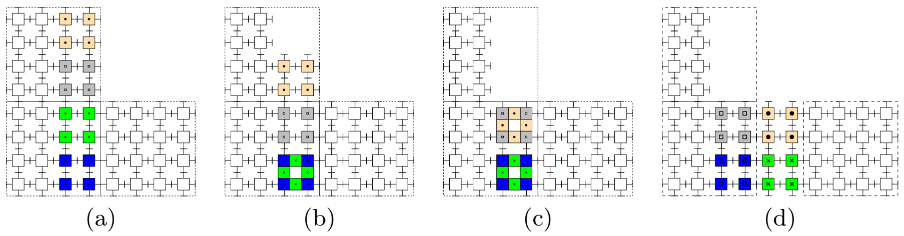

Fig. 8. *Parallel tunnel move. Three* 4×4 *modules are involved.*

4 Reconfiguration

In this section we show how to reconfigure a given robot to a canonical form with $O(\log n)$ parallel steps. Here we assume that the initial and final configurations of the robot consist of blocks of 32×32 atoms. However we will split blocks to use modules of 4×4 atoms in the intermediate configurations. Recall that the boundary has a width of four atoms.

Our divide-and-conquer algorithm proceeds as follows. Let the initial robot be placed on a grid of unit blocks (of 32×32 atoms). On this grid we construct a minimal square cell of side length 2^h that contains the initial robot (length is measured in block units). We recursively divide the cell into four subcells of length 2^{h-1}. As a base case, we take subcells of 2×2 blocks (i.e., containing 16×16 module lattice positions).

In parallel, we reconfigure each subcell within the same recursive depth, so that the resulting shape is easy to handle. Thus, by merging subcells, in $O(\log n)$ iterations we will have created a simple shape in our original square. Consider a cell M. We will use the inductive hypothesis that after merging its subcells,

M will become a ring if there are enough modules, or sparse otherwise. Furthermore, if two points on the boundary of M were initially connected, the new configuration will ensure connectivity via the shortest path through its boundary.

In the base case of our induction, M has length 2. Thus we have to merge four subcells, each of which is empty or full. We will obtain a ring if there is at least one full subcell. One such subcell contains 64 modules, which suffice to cover the boundary of M. Reconfiguration can be done by tunneling each interior module iteratively (or by the lemmas that will follow). Thus our hypothesis is preserved.

Lemma 1 *Consider a cell M. If any subcell of M contained a backbone in the original configuration, then there are enough modules to create a ring in M. There are also enough modules if a path originally connected two subcell sides that belong to the boundary of M but are not adjacent.*

Proof. Consider the eight exterior sides of subcells of M as shown in Fig. 9(a). Let each of the sides M_i have length c (i.e., c modules fill the side of a subcell). The total number of modules in the boundary of M is $8c-4$. A subcell backbone contains at least $8c$ modules and therefore suffices to cover the boundary.

Without loss of generality, suppose that a path begins on M_1 and ends at any side other than M_1, M_8, M_2. Then we have enough modules to make a ring in M, by similar counting as above. In fact to avoid having enough modules, such a path would have to remain within the lower two subcells. □

Lemma 2 *Let S_1 and S_2 be adjacent sparse subcells at the top of cell M. In the original robot, there can be no path from the top border of M to the other subcells (see Fig. 9(b)).*

Proof. A path from the top to the middle of M in the initial robot would contain enough modules to make both S_1 and S_2 rings. By the pigeon-hole principle, one of the two subcells cannot be sparse. □

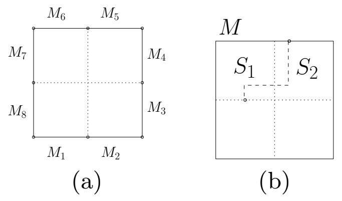

Fig. 9. *Connectivity issues, in Lemmas 1 and 2.*

Lemma 3 *All side-branches along the common border of two cells that are rings or sparse can be merged into at most two pieces per side, with $O(1)$ moves. Furthermore each side-branch touches one end of the border.*

Proof. If one cell is a ring then the other side can use it as a platform for a parallel tunnel move that will merge its side-branches into one piece. Otherwise, for each connected component of side-branches (of which there are at most two; one per corner) do the following.

Denote the two sides of the border by A and B. Absorb as much as possible from A to B by sliding modules from A across the border into vacant module lattice positions. Thus the component has one side-branch in B. Shift (parallel tunnel) the remainder of A towards the corner that the connected component attaches to, using B as a platform. Thus A becomes one side-branch. Now (either by a pop or by parallel-tunnel) bring back material from B to A to restore the original numbers in each cell. Thus each connected component consists of at most one side-branch from A and one from B. □

Lemma 4 *Suppose B is a boundary side of a cell that has been processed according to Lemma 3. Let A be a branch that is in the near-boundary adjacent to B, and has no connectivity purpose. We can absorb A into B, or B can be filled, with $O(1)$ moves.*

Proof. By Lemma 3, B contains at most two side-branches, each attached to a corner. If no modules in B are adjacent to A, we can use a 1-tunnel to move one node (endpoint) of A into the position in B that is adjacent to the other node of A. Then the rest of A can slide into B. Otherwise, if A is adjacent to a side-branch in B, as in Fig. 10(a), we do the following. Absorb parts of A into empty positions of B, as in Fig. 10(b). Thus we create a side-branch B_1 which can be used as a platform to be extended by performing a parallel tunnel move on what remains of A. If the extension causes B_1 to reach a corner or join to another side-branch in B, then B is full; see Fig. 10(c). □

For sparse cells, by repeatedly applying Lemma 4 and staircasing the remainder of A to the near-boundary side adjacent to B, we obtain the following:

Corollary 5 *If a branch A is positioned in the near-boundary of a sparse cell, either A can be fully absorbed into the boundary, or the cell will become a ring.*

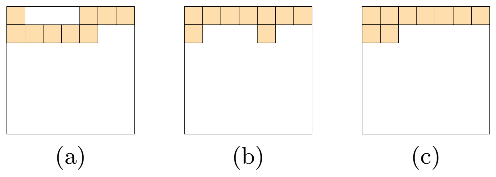

Fig. 10. *Absorbing a near-boundary branch into the boundary of a cell.*

Let a *merged cell* contain four subcells that satisfy our induction hypothesis. That is, they are either rings or sparse, and connectivity is ensured via shortest paths along their boundaries. A merged cell becomes *well-merged* if it is reconfigured to satisfy the induction hypothesis.

Lemma 6 *Let M be a merged cell containing three or four subcell rings. Then M can become a ring using $O(1)$ moves. Thus M becomes well-merged.*

Proof. Omitted due to space restrictions.
Sketch: The outer structure of the desired ring is either in place or can be completed easily. Following this, all that remains is to organize/merge the interior modules of the subcells. □

Lemma 7 *If exactly two subcells of a merged cell M are rings, then M can become well-merged using $O(1)$ moves.*

Proof. If the two sparse subcells are adjacent, then there is no critical connectivity maintained through their common border, by Lemma 2.

Apply Corollary 5 to move side-branches in the sparse subcells to the boundary of M. There is only one module that possibly cannot be moved, in the case of two rings that exist in a diagonal configuration and must be connected. If a new ring is created, we apply Lemma 6. Now the only branches along interior borders of subcells belong to the two rings, with the possible exception of one module at the middle of M. We can use corner pops and/or staircase moves and Corollary 5 to move the interior ring sides to the boundary of M while maintaining connectivity. This happens regardless of the relative position of the rings or the presence of the extra module.

What remains is to maintain our shortest path requirement, if we still do not have a ring in M. In this case, by Lemma 1 we know that there was no initial backbone in M. Thus each connected component of robot within M "covers" at most one corner (in other words there is at least one module gap per side).

Note that the modules in the two subrings alone nearly suffice to create a ring in M. Four modules are missing. We can remove a strip of width 2 from positions where we wish to have a gap in the boundary of M, and use parallel-tunneling to position this material in the current gaps. Essentially we create a temporary ring of width 2. Then the remaining material can be moved. □

Lemma 8 *If exactly one subcell S of a merged cell M is a ring, then M can become well-merged using $O(1)$ moves.*

Proof. Without loss of generality let S be at the bottom-left of M. By Lemma 2, in the original robot there was no path from the top border of M leading to either of the bottom subcells. The same holds for the right border of M and the two left subcells. Therefore the two interior borders between the three sparse subcells do not preserve any connectivity. We may use Corollary 5 to move branches from those interior borders to the boundary of M. Finally we can do the same for the interior sides of S.

We may have to redistribute excess internal material from within S. If M has become a ring, this is easy and has been discussed previously. Otherwise, we can apply Corollary 5 to each full row of the internal ring structure. This can be required at most eight times before a ring is created.

Our shortest path connectivity requirement is preserved directly, by the fact that the internal borders where not necessary for connectivity. □

Lemma 9 *If no subcell of a merged cell M is a ring, then M can become well-merged using $O(1)$ moves.*

Proof. By Lemma 2, we know that in the original robot configuration no path existed from a side of M to either of the two subcells furthest from it. Therefore all disjoint subgraphs maintained connectivity between at most two adjacent external sides of subcells. More specifically, the first type of allowed path connects points that are separated by a corner of M but are also inside the same subcell. By induction we assume that these points are already connected along the external boundary of their subcell. The second type connects points that are on the same border side of M (possibly adjacent subcells). Again by induction we know that they are already connected along the boundary of M. Therefore our shortest path requirement is preserved.

All that remains is to remove excess material from inner borders of subcells. This material consists of one or two branches per border, each of which is connected to the boundary of M. These can be staircased and redistributed with our standard procedures. □

Theorem 10 *Any source robot S can be reconfigured into any target robot T with $O(n \log n)$ atom operations in $O(\log n)$ parallel steps, if S and T are constructed with blocks of 32×32 atoms.*

Proof. Every cell retains the modules that it initially contained and does not interfere with the configuration of the robot outside the cell, until it is time to merge with its neighbors. A temporary exception to this occurs during Lemma 3. Therefore that step should be performed in a way so that no interference occurs (i.e., perform only this operation during one time step). At every time step, we merge groups of four cells, which by induction are either rings or sparse. By Lemmas 6–9, these four cells merge into a ring or sparse cell. Thus we construct a ring or sparse cell in $O(\log n)$ parallel time steps.

We show that the total number of operations is $O(n \log n)$. Each subcell containing m atoms can involve $O(m)$ parallel operations per time step. Because there are $O(1)$ time steps per level in the recursion, and all m_i sum to n, the total number of operations per recursion level is $O(n)$.

Now consider the bounding box B of S. We construct the smallest square B_2 of side length 2^h that contains S and has the same lower-left corner as B. Our recursive algorithm takes place within B_2. Now consider the last merge of subcells in our algorithm. The lower-left subcell L could not have contained S, because this would imply that $B_2 = L$. Therefore there must have been a path in S from the left side of B_2 leading to the two rightmost subcells (or from bottom to two topmost). This implies that S will become a ring (not sparse).

Because a ring of specific side length has a unique shape as a function of the number of modules it contains, the resulting ring in B_2 serves as a canonical form between S and T. □

5 Discussion

The number of atoms in our modules and initial blocks can be reduced. By using 2×2 modules instead of 4×4, some of our basic operations become relatively complicated. For example, the staircase move cannot be implemented via sliding, but instead involves a form of parallel tunneling to break off strips that are one module wide, and then using those as carrying tools, etc. Corner pops also become particularly unattractive. Reducing the block size has the result that we can no longer rely only on rings and sparse cells to maintain the connectivity of any robot. We obtain a small set of orthogonal shortcut trees that must be taken into consideration when merging cells. We conjecture that reconfiguration can take place with 2×2 modules and no block restriction.

Our algorithm seems to be implementable in $O(n \log n)$ time. Each subcell contains a constant number of rectangular components, so determining their relative configuration and series of motions should require constant time. We also claim that our results extend to the case of labeled robots. This would involve a type of merge-sort using staircase moves, once a straight path of modules is constructed using our algorithm.

We have not determined if a similar result will hold in 3D, or if $O(\log n)$ steps are optimal. Such a lower bound can be given for labeled robots, by a simple Kolmogorov argument: there exist permutations that contain $\Theta(n \log n)$ bits of information. Each parallel move can be encoded in $O(n)$ bits (for each robot in order, which sides perform which operations), so we need $\Omega(\log n)$ steps.

References

1. G. Aloupis, S. Collette, M. Damian, E. D. Demaine, D. El-Khechen, R. Flatland, S. Langerman, J. O'Rourke, V. Pinciu, S. Ramaswami, V. Sacristán, and S. Wuhrer. Realistic reconfiguration of telecube and crystalline robots. In *The Eighth International Workshop on the Algorithmic Foundations of Robotics (to appear)*, 2008.
2. G. Aloupis, S. Collette, M. Damian, E. D. Demaine, R. Flatland, S. Langerman, J. O'Rourke, S. Ramaswami, V. Sacristán, and S. Wuhrer. Linear reconfiguration of cube-style modular robots. *Computational Geometry - Theory and Applications (to appear; Preliminary version in Proc. 18th International Symposium on Algorithms and Computation (ISAAC2007), LNCS 4935, pp. 208-219, Springer-Verlag, 2007).*
3. Z. Butler, R. Fitch, and D. Rus. Distributed control for unit-compressible robots: Goal-recognition, locomotion and splitting. *IEEE/ASME Trans. on Mechatronics*, 7(4):418–430, 2002.
4. Z. Butler and D. Rus. Distributed planning and control for modular robots with unit-compressible modules. *Intl. Journal of Robotics Research*, 22(9):699–715, 2003.
5. D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, 2001.
6. J. W. Suh, S. B. Homans, and M. Yim. Telecubes: Mechanical design of a module for self-reconfigurable robotics. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 4095–4101, 2002.
7. S. Vassilvitskii, M. Yim, and J. Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 117–122, 2002.