Postprint: VLSI Array Reconfiguration Algorithm Based on Shortest Long-Link Priority Selection

Authors: Mò Fúhào, Qian Junyan, Qian Junyan

Date: 2022-11-02T00:00:00+00:00

Abstract

To improve the reconfiguration efficiency of very-large-scale integrated logic arrays, this paper proposes an improved algorithm based on the shortest long-link first selection principle. The algorithm constructs logic columns from both ends of the array respectively until the two logic columns intersect, at which point the construction terminates. Within the local region bounded by the two intersecting logic columns, the processor unit with the shortest long-link in each row is identified in a top-to-bottom manner, and the selected processor units are utilized to construct locally optimal logic columns. Building upon these operations and employing a divide-and-conquer approach, the newly obtained logic columns serve as boundaries for new local regions, and new locally optimal logic columns are acquired iteratively in sequence. Finally, by connecting the obtained locally optimal logic columns, the final target array can be obtained. The efficiency of the algorithm is verified through comparative analysis with existing reconfiguration algorithms. Simulation results demonstrate that, under the condition of maintaining constant logic array size, the proposed algorithm can effectively reduce the number of processor accesses during array reconfiguration compared to existing algorithms, and can reduce reconfiguration runtime to a certain extent, thereby improving the reconfiguration efficiency of logic arrays.

Full Text

Abstract

To improve the reconstruction efficiency of ultra-large-scale integrated logic arrays, this paper proposes an improved algorithm based on the shortest long-link priority selection principle. The algorithm constructs logical columns from both ends of the array, searching for processor units with the shortest long-links

in each row through intersecting logical columns. When intersecting logical columns are found, the construction process stops within a local area bounded by these columns. The selected processing units within this region are then used to construct locally optimal logical columns. Employing a top-down divide-and-conquer strategy, the algorithm connects these locally optimal logical columns, using each newly obtained logical column as the boundary for a new local region. Through iterative application of this process, new locally optimal logical columns are obtained sequentially until the final target array is constructed.

Comparative analysis and simulation results demonstrate that the proposed algorithm effectively reduces processor access count during array reconstruction and decreases reconstruction runtime while maintaining array scale. Compared with existing reconstruction algorithms, this approach significantly improves logic array reconstruction efficiency. The algorithm' s effectiveness is verified through comprehensive testing across various fault rates and array sizes.

Keywords: reconfiguration; algorithm; array; fault tolerance; high-efficiency

1 VLSI Array Structure and Problem Description

Consider an original array H of size m×n, where m and n represent the number of rows and columns respectively. Let p denote the fault density of the original array, and N represent the number of faulty processing units. A faulty processing unit is defined as one that cannot process data or receive data from other processing units. After reconfiguration, the resulting array is called a logical array T of size m' ×n' , which contains no faulty processing units. Rows in the original and logical arrays are referred to as physical rows and logical rows, respectively.

The physical architecture of the array is illustrated in [FIGURE:N]. Each switch has multiple link states, and each processing unit can change its connection pattern by altering the routing switch states. For a processing unit e_{i,j} where i is the row index and j is the column index, there are two basic reconfiguration schemes: row bypass and column rerouting. In row bypass, a faulty processing unit e_{i,j} is bypassed, while column rerouting follows a similar definition. In column rerouting, if e_{i,j} is faulty, it can be directly connected to e_{i+1,j} through external switches. The compensation distance d is defined to ensure low-power hardware implementation.

For each fault-free processing unit in a row, the upper and lower adjacent node sets are denoted as A_{up} and A_{down}, defined as follows:

A_{up}(u) = {v | v is fault-free, row(v) - row(u) = 1, |col(v) - col(u)| ≤ 1, 2 ≤ row(v) ≤ m}

A_{down}(u) = {v | v is fault-free, row(v) - row(u) = 1, |col(v) - col(u)| ≤ 1, 1 ≤ row(v) ≤ m-1}

For any processing unit v in A_{down}(u), v is called the lower-left adjacent

processing unit of u if col(v) - col(u) = -1, the lower-middle adjacent unit if col(v) - col(u) = 0, and the lower-right adjacent unit if col(v) - col(u) = 1.

Link connections are classified as short connections or long connections based on the number of switches used. A connection using n switches to connect processing units in the target array is called a long connection. A two-dimensional logic array has multiple possible link patterns. The research problem involves finding a fault-free high-performance target array (HPTA) with the minimum number of long links in a mesh-connected processor array with faulty nodes, under row bypass and column rerouting constraints.

2 ACRIL Algorithm

2.1 Algorithm Overview

The ACRIL (Acceleration algorithm using shortest long-link priority selection) algorithm employs a divide-and-conquer strategy to solve the reconstruction problem without limiting compensation distance. The overall approach is as follows:

During each iteration, the algorithm constructs logical columns from left to right and right to left simultaneously, forming intersecting logical columns. The intersection points are stored in a set. Using an intelligent search algorithm, three sub-processes—Up_{Process}, Down_{Process}, and Shest_{Process}—are executed to obtain path segments. These three path segments are then merged to form a complete logical column. The newly obtained logical column serves as a new boundary, and the process continues iteratively until all logical columns are constructed.

The algorithm reduces access to fault-free nodes during logical column reconstruction, thereby improving reconstruction efficiency and reducing reconstruction time for the original array.

2.2 Sub-process Descriptions

Up_{Process}: Solves for the sub-logical column from the first common processing unit to the first row.

Down_{Process}: Solves for the sub-logical column from the last common processing unit to the last row.

Shest_{Process}: Solves for the sub-logical column between common processing units.

The pseudo-code for finding the optimal logical column is as follows:

```
while intersection set is not empty:
    // Process first row element
    heuristic search algorithm to find Up_{Process}
```

```
    // Process last row element
    heuristic search algorithm to find Down_{Process}

    // Process intersection points
    for each pair of adjacent intersections:
        directly add to path
    heuristic search algorithm to find Shest_{Process}

    // Merge paths
    merge three path segments to form complete logical column
```

The Shest_{Process} sub-process employs heuristic search principles. For a path P between two common nodes S and T, the algorithm calculates node weights and maintains open and closed lists. Nodes are added to appropriate lists based on specific conditions. When a node is already in the open list, its weight is recalculated. The process terminates when the open list is empty or the current node equals T.

The pseudo-code for the sub-process is:

```
while open list is not empty:
    get node u with minimum weight from open list as current node
    add current node to closed list

    if current node is T:
        return path P between S and T

    while lower adjacent set of current node is not empty:
        get lower adjacent node u
        calculate weight of u

        if u in open list:
            recalculate weight of u
        else:
            add u to open list

        remove u from lower adjacent set
```

The path segments obtained from the three sub-processes are merged to form a complete logical column. The node access count for ACRIL is significantly lower than that of existing algorithms, as each sub-process primarily utilizes heuristic search principles to minimize unnecessary node visits.

3 Experimental Analysis

To validate the effectiveness of the ACRIL algorithm, we implemented it and compared its performance with existing reconstruction algorithms. All algorithms were executed in the same experimental environment: an Intel processor

with Windows operating system.

The experiments tested various array sizes (32$×32, 48×48, 64×$64) with fault rates set to 1%, 5%, 10%, and 15%. For each configuration, we measured reconstruction time and node access count.

3.1 Performance Metrics

[TABLE:N] presents the performance metrics of the ACRIL algorithm compared with existing methods. The results show that under identical fault rate conditions, both algorithms obtain target logical arrays of the same scale, but ACRIL achieves shorter reconstruction times, indicating higher efficiency in recovering from fault states.

In a 48$×48$originalarraywitha$5×$64 array with the same fault rate, existing algorithms take 892.4 seconds, whereas ACRIL completes reconstruction in 766.3 seconds, yielding a 14.13% efficiency improvement.

3.2 Node Access Comparison

The experimental data demonstrates that ACRIL significantly reduces node access count during logical column reconstruction compared to existing algorithms. This reduction in processor unit access frequency directly contributes to decreased reconstruction time and lower runtime performance overhead.

For a 32$×32$originalarray, thenodeaccesspatternsareasfollows : −At1×31targetarray−At5×27targetarray − At10×$23 target array

Despite producing identical target array scales, ACRIL accesses fewer nodes during the reconstruction process, confirming its superior efficiency.

4 Conclusion

This study proposes an improved array reconstruction algorithm based on shortest long-link priority selection. By reducing processing unit access count during reconstruction, the algorithm accelerates the reconfiguration process and improves overall efficiency. Experimental results confirm that ACRIL outperforms existing reconstruction algorithms across various array sizes and fault rates, reducing both node access count and reconstruction time.

Future research will focus on developing new improvements that consider the impact of logical column selection on array scale during the construction process, particularly for degradable arrays, to provide further performance optimization for array reconstruction.

References

[1] ZHANG Li. Fault-tolerant meshes with small degree[J]. IEEE Transactions on Computers, 2002, 51(5): 553-560.

[2] KUO Y, CHEN Y. Efficient reconfiguration algorithms for degradable VLSI/WSI arrays[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1992, 11(10): 1289-1300.

[3] LOW C P, LEONG H W. On the reconfiguration of degradable VLSI/WSI arrays[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1997, 16(10): 1213-1221.

[4] XIANG Dong, CHAKRABARTY K, FUJIWARA H. Fault-tolerant design of network-on-chip for unicast and multicast communications[J]. ACM Transactions on Design Automation of Electronic Systems, 2018, 23(73): 1-73.

[5] XIANG Dong, PAN Y. Reliable routing algorithm based on unicast and multicast testing for network-on-chip[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019, 1(6): 92-110.

[6] CAI Yuan, XIANG Dong, JIANG Jigang. Low-power adaptive routing algorithm for high-performance network-on-chip[J]. IET Transactions on Computers and Digital Techniques, 2019, 14(11): 1783-1792.

[7] WU Jigang, SRIKANTHAN T, JIANG Guiyuan. Accelerating reconfiguration algorithm for degradable arrays[J]. IEEJ Transactions on Electrical and Electronic Engineering, 2018, 13(10): 1511-1519.

[8] QIAN Junyan, CHEN Cong, ZHAO Lingzhong, et al. Reconfiguring arrays for maximum power efficiency satisfiability[J]. IEEJ Transactions on Electrical and Electronic Engineering, 2018, 13(5): 770-776.

[9] QIAN Junyan, HUANG Bisheng, DING Hao, et al. An efficient multiple shortest augmenting paths algorithm for constructing high-performance subarray[J]. Integration, the VLSI Journal, 2021, 75: 63-72.

[10] WU Jigang, SRIKANTHAN T, JIANG Guiyuan. Constructing sub-arrays with short interconnects[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(4): 929-938.

[11] XIAO Hanpeng, QIAN Junyan, ZHOU Zhide, et al. Integer programming model for tightly-coupled processor array reconfiguration[J]. Journal of Guilin University of Electronic Technology, 2018, 38(1): 61-64.

[12] HUANG Bisheng, DING Hao, QIAN Junyan, et al. Efficient reconfiguration algorithm for subarray based on network flow[J]. Journal of Guilin University of Electronic Technology, 2019, 39(6): 466-470.

[13] DING Hao, QIAN Junyan, ZHAO Lingzhong, et al. Maximum target array reconfiguration algorithm based on integer programming[J]. Journal of Guilin University of Electronic Technology, 2019, 39(6): 466-470.

[14] DING Yiping, CHANG Liang, et al. Reconfiguring subarrays under row and column rerouting[J]. IEEE Access, 2017, 5: 23912-23919.

[15] QIAN Junyan, WANG Dong, JIANG Guiyuan. Mathematical model for reconfiguring two-dimensional mesh-connected subarrays under row and column rerouting[J]. Journal of Guilin University of Electronic Technology, 2017, 37(6): 458-462.

[16] DING Hao, QIAN Junyan, ZHAO Lingzhong, et al. Flexible scheme for reconfiguring subarrays under row and column rerouting[J]. Parallel Computing, 2021, 151: 1-12.

[17] QIAN Junyan, ZHOU Zhide, GU Tianlong. Optimal reconfiguration of high-performance subarrays with network flow[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(12): 3575-3587.

[18] QIAN Junyan, DING Hao, XIAO Hanpeng. Efficient reconfiguration algorithm for constructing sub-arrays under flexible rerouting schemes[J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(1): 36-39.

[19] BAI Zhangshun, QIAN Junyan, ZHAO Lingzhong, et al. Abstract model for fault-tolerant processor array reconfiguration[J]. Journal of Guilin University of Electronic Technology, 2017, 37(1): 40-43.

[20] HU Jia, QIAN Junyan, ZHAO Lingzhong, et al. High-performance array reconfiguration method[J]. Journal of Guilin University of Electronic Technology, 2017, 37(1): 36-39.

[21] JIANG Guiyuan, WU Jigang, SUN Jizhou, et al. Reconfiguration algorithms for three-dimensional arrays[J]. IEEE Transactions on Computers, 2015, 64(10): 2926-2939.

[22] JIANG Guiyuan, WU Jigang, SUN Jizhou, et al. Flexible reconfiguration schemes for three-dimensional processor arrays[J]. Journal of Parallel and Distributed Computing, 2014, 74(10): 3026-3036.

[23] JIANG Guiyuan, WU Jigang, SUN Jizhou, et al. Reconfiguration of mesh-connected multiprocessor arrays with reduced interconnection length[C]//International Conference on Parallel Architectures and Algorithms. Berlin, German: Springer, 2014: 497-510.

Note: Figure translations are in progress. See original paper for figures.