Journal of Systems Architecture 75 (2017) 15–25

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

An efficient data structure for dynamic two-dimensional reconfiguration☆,☆☆

Sándor P. Fekete*, Jan-Marc Reinhardt, Christian Scheffer

Department of Computer Science, TU Braunschweig, Germany

ARTICLE INFO

Article history:
Received 4 July 2016
Revised 30 January 2017
Accepted 17 February 2017
Available online 27 February 2017

Keywords:
FPGAs
Partial reconfiguration
Two-dimensional reallocation
Defragmentation
Dynamic data structures
Insertions and deletions

ABSTRACT

In the presence of dynamic insertions and deletions into a partially reconfigurable FPGA, fragmentation is unavoidable. This poses the challenge of developing efficient approaches to dynamic defragmentation and reallocation. One key aspect is to develop efficient algorithms and data structures that exploit the two-dimensional geometry of a chip, instead of just one. We propose a new method for this task, based on the fractal structure of a quadtree, which allows dynamic segmentation of the chip area, along with dynamically adjusting the necessary communication infrastructure. We describe a number of algorithmic aspects, and present different solutions. We also provide a number of basic simulations that indicate that the theoretical worst-case bound may be pessimistic.

1. Introduction

In recent years, a wide range of methodological developments on FPGAs aim at combining the performance of an ASIC implementation with the flexibility of software realizations. One important development is partial runtime reconfiguration, which allows overcoming significant area overhead, monetary cost, higher power consumption, or speed penalties (see e.g. [2]). As described in [3], the idea is to load a sequence of different modules by partial runtime reconfiguration.

In a general setting, we are faced with a dynamically changing set of modules, which may be modified by deletions and insertions. Typically, there is no full a-priori knowledge of the arrival or departure of modules, i.e., we have to deal with an online situation. The challenge is to ensure that arriving modules can be allocated. Because previously deleted modules may have been located in different areas of the layout, free space may be fragmented, making it necessary to *relocate* existing modules in order to provide sufficient area. In principle, this can be achieved by completely *defragmenting* the layout when necessary; however, the lack of control over the module sequence makes it hard to avoid frequent full defragmentation, resulting in expensive operations for insertions if a naïve approach is used.

Dynamic insertion and deletion are classic problems of Computer Science. Many data structures (from simple to sophisticated) have been studied that result in low-cost operations and efficient maintenance of a changing set of objects. These data structures are mostly one-dimensional (or even dimensionless) by nature, making it hard to fully exploit the 2D nature of an FPGA. In this paper, we propose a 2D data structure based on a quadtree for maintaining the module layout under partial reconfiguration and reallocation. The key idea is to control the overall structure of the layout, such that future insertions can be performed with a limited amount of relocation, even when free space is limited.

Our main contribution is to introduce a 2D approach that is able to achieve provable constant-factor efficiency for different types of relocation cost. To this end, we give detailed mathematical proofs for a slightly simplified setting, along with sketches of extensions to the more general cases. We also provide basic simulation runs for various scenarios, indicating the quality of our approach.

The rest of this paper is organized as follows. The following Section 2 provides a survey of related work. For better accessiblity of the key ideas and due to limited space, our technical description in Section 3, Section 4, and Section 5 focuses on the case of discretized square modules on a quadratic chip area. We discuss in Section 6 how general rectangles can be dealt with, with corresponding simulations in Section 7. Along the same lines, we do not explicitly elaborate on the dynamic maintenance of the

☆ This work was supported by the DFG Research Group FOR-1800, "Controlling Concurrent Change", under contract number FE407/17-1 and 17-2.

☆☆ A preliminary extended abstract of this paper appears in ARCS2016 [1].

* Corresponding author.
E-mail addresses: s.fekete@tu-bs.de (S.P. Fekete), j-m.reinhardt@tu-bs.de (J.-M. Reinhardt), scheffer@ibr.cs.tu-bs.de (C. Scheffer).

http://dx.doi.org/10.1016/j.sysarc.2017.02.004
1383-7621/

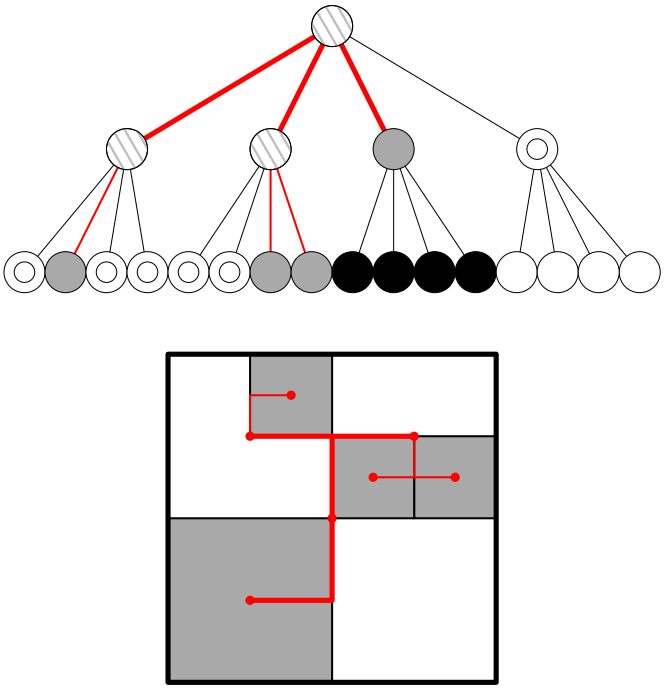

Fig. 1. A quadtree configuration (above) and the corresponding dynamically generated quadtree layout below). Gray nodes are occupied, white ones with gray stripes fractional, black ones blocked, and white nodes without stripes empty. Maximally empty nodes have a circle inscribed. Red lines in the module layout indicate the dynamically produced communication infrastructure, induced by the quadtree structure. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

communication infrastructure; see Fig. 1 for the basic idea. Further details are left to future work, with groundwork laid in [4].

2. Related work

The problem considered in our paper has a resemblance to one-dimensional *dynamic storage allocation*, in which a sequence of storage requests of varying size have to be assigned to a block of memory cells, such that the length of each block corresponds to the size of the request. In its classic form (without virtual memory), this block needs to be contiguous; in our setting, contiguity of two-dimensional allocation is a must, as reconfigurable devices do not provide techniques such as paging and virtual memory. Once the allocation has been performed, it is static in space: after a block has been occupied, it will remain fixed until the corresponding data is no longer needed and the block is released. As a consequence, a sequence of allocations and releases can result in fragmentation of the memory array, making it hard or even impossible to store new data.

On the practical side, classic buddy systems partition the one-dimensional storage into a number of standard block sizes and allocate a block in a smallest free standard interval to contain it. Differing only in the choice of the standard size, various systems have been proposed [5–9]. Newer approaches based on cache-oblivious structures in memory hierarchies include Bender et al. [10,11]. Theoretical work on one-dimensional contiguous allocation includes Bender and Hu [12], who consider maintaining n elements in sorted order, with not more than $O(n)$ space. Bender et al. [13] aim at reducing fragmentation when maintaining n objects that require contiguous space. Fekete et al. [3] study complexity results and consider practical applications on FPGAs. Reallocations have also been studied in the context of heap allocation. Bendersky and Petrank [14] observe that full compaction, i.e., creating a contiguous block of free space on the heap, is prohibitively expensive and consider partial compaction. Cohen and Petrank [15] extend these to practical applications. Bender et al. [16] describe a strategy that achieves good amortized movement costs for reallocations, where allocated blocks can be moved at a cost to a new position that is disjoint from with the old position. Another paper by the same authors [17] deals with reallocations in the context of scheduling. Examples for packing problems in applied computer science come from allocating FPGAs. Fekete et al. [18] examined a problem dealing with the allocation of different types of resources on an FPGA that had to satisfy additional properties. For example, to achieve specified clock frequencies diameter restrictions had to be obeyed by the packing. The authors were able to solve the problem using integer linear programming techniques.

Over the years, a large variety of methods and results for allocating storage have been proposed. The classical sequential fit algorithms, First Fit, Best Fit, Next Fit and Worst Fit can be found in Knuth [19] and Wilson et al. [20]. These are closely related to problems of offline and online packing of two-dimensional objects. One of the earliest considered packing variants is the problem of finding a dense packing of a known set of squares for a rectangular container; see Moser [21], Moon and Moser [22] and Kleitman and Krieger [23], as well as more recent work by Novotný [24,25] and Hougardy [26]. There is also a considerable number of other related work on offline packing squares, cubes, or hypercubes; see [27–29] for prominent examples. The *online* version of square packing has been studied by Januszewski and Lassak [30] and Han et al. [31], with more recent progress due to Fekete and Hoffmann [32,33]. A different kind of online square packing was considered by Fekete et al. [34,35]. The container is an unbounded strip, into which objects enter from above in a Tetris-like fashion; any new object must come to rest on a previously placed object, and the path to its final destination must be collision-free.

There are various ways to generalize the online packing of squares; see Epstein and van Stee [36–38] for online bin packing variants in two and higher dimensions. In this context, also see parts of Zhang et al. [39]. A natural generalization of online packing of squares is online packing of rectangles, which have also received a serious amount of attention. Most notably, online strip packing has been considered; for prominent examples, see Azar and Epstein [40], who employ shelf packing, and Epstein and van Stee [36]. Offline packing of rectangles into a unit square or rectangle has also been considered in different variants; for examples, see [41], as well as [42]. Particularly interesting for methods for online packing into a single container may be the work by Bansal et al. [43], who show that for any complicated packing of rectangular items into a rectangular container, there is a simpler packing with almost the same value of items.

From within the FPGA community, there is a huge amount of related work dealing with problems related to relocation. Becker et al. [44] present a method for enhancing the relocability of partial bitstreams for FPGA runtime configuration, with a special focus on heterogeneities. They study the underlying prerequisites and technical conditions for dynamic relocation. Gericota et al. [45] present a relocation procedure for Configurable Logic Blocks (CLBs) that is able to carry out online rearrangements, defragmenting the available FPGA resources without disturbing functions currently running. Another relevant approach was given by Compton et al. [46], who present a new reconfigurable architecture design extension based on the ideas of relocation and defragmentation. Koch et al. [47] introduce efficient hardware extensions to typical FPGA architectures in order to allow hardware task preemption. These papers do not consider the algorithmic implications and how the relocation capabilities can be exploited to optimize module layout in a fast, practical fashion, which is what we consider in this paper. Koester et al. [48] also address the problem of