

GifT:A Gravity-Directed and Life-Time Based Algorithm for Temporal Partitioning of Data Flow Graphs

Mehdipour, Farhad

Computer and IT Engineering Department, Amirkabir University of Technology

Saheb Zamania, Morteza

Computer and IT Engineering Department, Amirkabir University of Technology

Sedighi, Mehdi

Computer and IT Engineering Department, Amirkabir University of Technology

Murakami, Kazuaki

Department of Informatics, Graduate School of Information Science and Electrical Engineering, Kyushu University

他

<https://hdl.handle.net/2324/6794492>

出版情報 : Proceedings of The 2006 International Conference on Engineering of Reconfigurable Systems & Algorithms (ERSA'06), pp.227-230, 2006-06. International Conference on Engineering of Reconfigurable Systems & Algorithms

バージョン :

権利関係 :



GifT: A Gravity-Directed and Life-Time Based Algorithm for Temporal Partitioning of Data Flow Graphs

Farhad Mehdipour, Morteza Saheb Zamani,
Mehdi Sedighi
Computer and IT Engineering Department,
Amirkabir University of Technology,
Tehran, Iran

Kazuaki Murakami, Hamid Noori
Department of Informatics, Graduate School of
Information Science and Electrical Engineering,
Kyushu University, Japan

Abstract

In reconfigurable systems, reconfiguration latency has a significant impact on the system performance. In this work, a temporal partitioning algorithm is presented to partition data flow graphs for reconfigurable computing systems. Life-time of a node in a data flow graph represents the number of times it executes during the application run. A new factor, called residing eligibility, inspired from the Universal gravitation law, is introduced to depict the eligibility of a node to stay in succeeding partitions and to prohibit it from being swapped in/out. Life-time, size, and distance of a node from the nearest identical node in the data flow graph are important factors for determining its residing eligibility. Assigning nodes to a partition according to their residing eligibility can cause fewer nodes with different functionalities to be assigned to subsequent partitions. Thus, reconfiguration overhead time and also wasted hardware space decreases due to common parts in subsequent configurations on a partially programmable hardware.

Keywords: Reconfigurable computing systems, Temporal partitioning, Data flow graph.

1. Introduction

Reconfigurable computing systems (RCS) are an alternative to application-specific integrated circuits (ASICs) and general-purpose processors [1, 17]. An RCS is a combination of a general-purpose processor and a reprogrammable hardware (like an FPGA¹). One of the main challenges in the reconfigurable computing domain is the lack of appropriate compilers. A static compiler is used before run-time to generate configurations and their scheduling. Static reconfiguration refers to having the ability to reconfigure a system, but not during the execution time [8]. To implement a large circuit on an FPGA, it may have to be partitioned into multiple stages. Each configuration can be swapped in/out to implement each

stage one by one and perform the functions of the original circuit. This type of partitioning is known as temporal partitioning [3,6].

Bobda [3] proposed a spectral placement to position the modules in a three-dimensional vector space. Karthikeya et al. [5] proposed algorithms for temporal partitioning and scheduling of large designs on area constrained reconfigurable hardware but did not consider the reconfiguration time overhead. *SPARCS* [11] is an integrated partitioning and synthesis framework, which has a temporal partitioning tool to temporally divide and schedule the tasks on a reconfigurable system. Luk et al. [7,13] proposed a methodology to take advantage of common operators in successive partitions. It attempted to reduce configuration time and thus, the application execution time. Tanougust et al. [15] attempted to find the minimum area while meeting timing constraints. In [14], Spillane and Owen focused on finding a sequence of conditions for activating an appropriate component at a particular time and optimizing successive configurations to achieve the desired trade-offs among reconfiguration time, operation speed and area. In [9], a similarity-based partitioning algorithm was proposed, which finds modules with the same functionality in a data flow graph (DFG) and then attempts to increase the similarity of adjacent configurations. This results in a shorter placement time for similar partitions on the target FPGA for the compilation process and also shorter reconfiguration time.

In this paper, we propose a novel temporal partitioning algorithm called **gravity-directed and life-time based algorithm** (*GifT*), which attempts to prohibit high cost swapping in/out of modules during run-time of application by using a new factor, named *residing eligibility*. Residing eligibility is calculated based on the life-time and the inertia of modules according to Universal gravitation law. Life-time of a node represents how many times it appears in DFG. Inertia or mass of a module represents the size of the corresponding node in DFG. This algorithm intends to reduce the reconfiguration overhead time and is used at the design time where there is enough time to explore the search space and to get close to the optimality of design criteria. We explain the *GifT* algorithm, and the

¹- Field Programmable Gate Array

idea behind it in Section 2. In Section 3, *Gift* parameters and the way to calculate them are presented. In Section 4, the details of our implementation are explained and experimental results are presented and finally, Section 5 concludes the paper.

2. *Gift* Temporal Partitioning

Temporal partitioning can be stated as partitioning a data flow graph (DFG) into a number of partitions such that each partition can fit in the target device and also, dependencies among the graph nodes are not violated. For a partially-reconfigurable hardware, parts of the hardware can be programmed without disturbing the rest of the design since common parts of two successive configurations can remain unchanged. Our main goal is to reduce the reconfiguration time and overall run-time of applications. We assume that the target programmable device is partially programmable. The proposed algorithm takes a DFG, the nodes of which represent pre-designed modules in a library. For large modules of DFG, which may occur in succeeding configurations, swapping them in/out may not be cost-effective, since it increases reconfiguration overhead time. Therefore, it is reasonable to assign large modules with more inertia and identical nodes at closer distance to subsequent partitions. From this theoretical fact, *Gift* algorithm was inspired from the *Universal gravitation law*: Newton came to the conclusion that every object in the Universe attracts every other object with a force proportional to the product of their masses and inversely proportional to the square of the distance between the two objects.

The main idea behind *Gift* is considering potential of DFG nodes to stay at the subsequent partitions with respect to their priority. Priority of DFG nodes is calculated according to their life time and inertia. Using this idea, some modules of current configuration are replicated at the next configuration without need for using them during run-time. This brings about smaller bit-stream size for the next configuration. According to our methodology, identical nodes in a DFG are the nodes that have the same functionality. For example, assume for a selected node and its identical node in input data flow graph, first and third partitions are the probable appropriate partitions to assign them. According to the *Gift* idea, if a node is replicated in second partition with no need to be run, this causes that the parts of programmable device, which have been allocated to this node remain unchanged at the run-time. Therefore, less reconfiguration time is needed to reconfigure second and third partitions. This effect will be more stressed if a larger size node is considered. In addition, distance of the identical nodes in DFG is an important factor which determines their attraction force exert each other according to Universal gravitation law. In other words, nodes those have non-partitioned closer

identical nodes in DFG will have the higher priority for partitioning than the other nodes with far or without identical nodes.

We define a new factor named residing eligibility (*RE*), which shows the eligibility of a module for staying in the next partition, whereas it has been partitioned in the current configuration. Higher *RE* value shows the high eligibility of a module for partitioning and staying at the next partition to decrease the effect of its inertia on reconfiguration time. A naïve approach for temporal partitioning is using the ASAP¹ scheduling algorithm [3, 10]. This algorithm schedules a data flow graph in an attempt to minimize latency by topologically sorting of the nodes of the graph [10]. In our proposed partitioning process, assigning priority of node for partitioning is done according to ASAP level of node in DFG, size of node and the distance of node with other partitioned identical nodes located in previous partitions. In other words, *RE* represents the priority of a node for partitioning. We consider two types of partitioned nodes in our methodology:

- I. The nodes which are not identical to any node in the non-partitioned nodes of the DFG.
- II. The nodes which have identical nodes in the non-partitioned nodes.

The life-time of a node represents the number of times it runs during the application run. Assume that a node is assigned to the current configuration. Both the life-time of the node and its distance to a similar unprocessed node in the DFG are important factors which represent the eligibility of the node to reside in the next configuration. Staying a node in the subsequent configuration prohibits it to be swapped in/out and therefore, it can reduce the reconfiguration time.

For the node i of type *I*, $RE(i)$ is defined as:

$$RE(i) = (1 - Level(i)/MaxLevel) + \epsilon * 1/(1 + Slack(i)) \quad (1)$$

where $Level(i)$ is the ASAP level of node i , $MaxLevel$ is the maximum levels of nodes, ϵ is a small number and $Slack(i)$ is the slack of node i in the DFG. The first term represents the importance of *ASAP* level of the node and the second term is for tie breaking. For the processed node i of type *II* with a similar node j in the remaining unprocessed nodes of the DFG, $RE(i, j)$ is defined as:

$$RE(i, j) = w_1 * (1 - Level(j)/MaxLevel) + w_2 * Size(i)/(MaxNodeSize * (Level(j) - Level(i) + 1)^2) + \epsilon * 1/(1 + Slack(i)) \quad (2)$$

where w_1 and w_2 are the weights of each term. The first term is similar to the first term of Equation (1). The second term represents the role of node size in the residing eligibility of the node. Size of a node is proportional to its mass and inertia. According to the Universal gravitation law, larger nodes with larger mass exert more attraction. On the other hand, the closer nodes, the more attraction force they exert on

¹ - As Soon As Possible

each other. Thus, for a processed node, which has a similar unprocessed node, its RE value determines whether it should reside on hardware or not.

As an example, consider the DFG of Figure 1 as the input to the algorithm. Table 1, shows the RE values at different rounds for the DFG in Figure 1. The nodes selected for each partition in every round are depicted in bold face. Nodes 11 and 15 are replicated in two subsequent partitions.

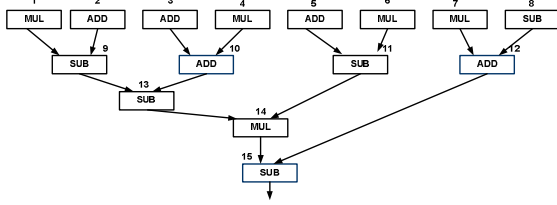


Figure 1: A sample DFG

Table 1. RE values at different rounds

Node	1 st Round	2 nd Round	3 rd Round
1	0.801		
2	0.801		
3	0.801		
4	0.801		
5	0.800		
6	0.800	0.912	
7	0.800	0.800	
8	0.800		
9	0.601		
10	0.601		
11	0.600	0.608	
12	0.600	0.609	
13	0.401	0.401	
14	0.201	0.201	0.201
15	0.001	0.001	0.001

3. Calculating w_1 and w_2

RE values are computed according to Equations (1) and (2) for the two types of processed nodes, respectively. Appropriate values for w_1 and w_2 should be calculated in such a way that legal values can be achieved for RE. To determine these coefficients, we assume that RE values are normalized ($0 < RE < 1$). For a node of type I, $RE(i) = (1 - Level(i)/MaxLevel) < 1$ (we ignore the last term that is dependent on ϵ). Also, for a node j of type II, which is similar to an unprocessed node k , $RE(j, k) = w_1 * (1 - Level(k)/MaxLevel) + w_2 * (Size(j)/(MaxNodeSize * (Level(k) - Level(j) + 1))^2) < 1$. In addition, if it is assumed that $Level(k) \leq Level(i)$, it will be obvious that: $RE(i) \leq RE(j, k)$. Therefore, the two following inequalities are obtained:

$$w_1 * (1 - Level(k)/MaxLevel) + w_2 * (Size(j)/(MaxNodeSize * (Level(k) - Level(j) + 1))^2) > (1 - Level(i)/MaxLevel) \quad (3)$$

$$w_1 * (1 - Level(k)/MaxLevel) + w_2 * (Size(j)/(MaxNodeSize * (Level(k) - Level(j) + 1))^2) < 1 \quad (4)$$

For the first inequality, it can be assumed that the maximum value of the right expression must be less than the minimum value of the left expression and so, the following inequality is obtained:

$$w_2 > ((1 - w_1) * (1 - Level(k)/MaxLevel) * MaxNodeSize * MaxLevel^2) / MinNodeSize \quad (5)$$

For the inequality (5), the maximum value of the left expression should be less than 1:

$$w_1 * (1 - 1/MaxLevel) + w_2 < 1 \quad (6)$$

By solving the linear programming problem obtained, which consists of two inequalities (5) and (6), w_1 and w_2 are calculated as follows (Figure 2):

$$w_1 = (X * Y - 1) / X * (Y + 1) \text{ and } w_2 = 1 - w_1 * X \quad (7)$$

$$X = 1 - 1/MaxLevel \text{ and}$$

$$Y = MaxNodeSize * MaxLevel^2 / MinNodeSize \quad (8)$$

According to (7) and (8), we find that w_1 and w_2 are entirely dependent on the characteristics of the input DFG and therefore, they should be calculated for the DFG at the initialization step of *GifT*.

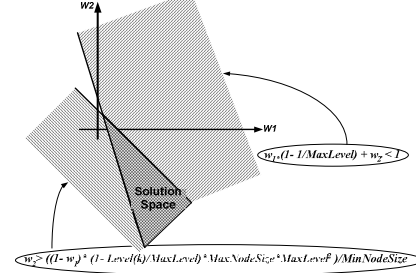


Figure 2. Solution space for calculating w_1 and w_2

4. Experimental Results

In our presented temporal partitioning algorithm a DFG is taken as the input. A library consisting of the required modules has been developed. Each module was described in *VHDL* and was then synthesized by *Leonardo Spectrum* synthesis tool to obtain a structural description. The *SIS* synthesis package [12] was used to perform technology-independent logic optimization of each module circuit. Next, each circuit was technology-mapped into 4-LUTs¹ and flip flops by *FlowMap* [4]. The output of *FlowMap* is a netlist of LUTs and flip flops in *.blif* format. *T-VPack* [3] then packed this netlist of 4-LUTs and flip flops into more coarse-grained logic blocks. We used *VPR* [3] that is one of the popular tools for placement and routing of the configurations. The architecture of the target programmable device was chosen to be a *Xilinx Virtex (XCV100)* FPGA. We chose five static data flow graphs from [3] and applied our tool to them.

We assume that the reconfiguration time can be approximated by a linear function of the total area of functional units being reconfigured. Usually, the run time of each configuration is much less than a microsecond, whereas the full reconfiguration of a programmable device is typically done in several microseconds. Therefore, reducing the reconfiguration time decreases the overall run time of the application accordingly. Experiments showed that *GifT* often results in more common CLBs in subsequent configurations comparing with greedy temporal

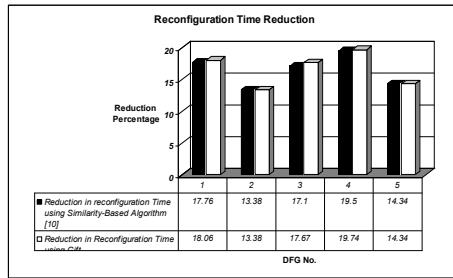
¹Four-inputs LookUp Table

partitioning algorithm presented in [9] (Table 2). Thus, more reduction in reconfiguration overhead time is achieved by *Gift* algorithm. These two algorithms generated the same number of partitions.

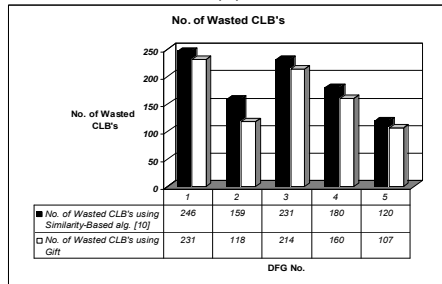
Figure 3(a) shows the improvement in reconfiguration time for both similarity-based algorithm [9] and *Gift*. *Gift* has usually more reduction in reconfiguration overhead time. Another result showed that *Gift* not only had no extra area overhead but also it reduced the wasted area in partitions. Figure 3(b) shows that the wasted number of CLB's using the *Gift* algorithm is less than that generated by the algorithm presented in [9].

Table 2. Comparison of *Gift* and the algorithm in [9].

DFG	DFG Size (No. of CLB's)	Similarity-Based algorithm [9]		<i>Gift</i>	
		No. of Partitions	No. of Similar CLB's	No. of Partitions	No. of Similar CLB's
1	1059	3	627	3	637
2	900	3	401	3	401
3	1080	4	616	4	636
4	1217	4	791	4	801
5	948	3	453	3	453



(a)



(b)

Figure 3. Reduction in reconfiguration time (a) reduction in wasted space (b)

5. Conclusion

In this work, a new temporal partitioning algorithm was proposed to generate configurations for a reconfigurable computing system. *RE* is a new factor, which defines the eligibility of a module for staying in

the subsequent partition to reduce reconfiguration overhead time. It is dependent on the life-time and the inertia of a module. Some of modules are replicated in the succeeding configurations, which brings about more common CLB's and reduction in the reconfiguration time without any extra area overhead and even with smaller wasted area.

Acknowledgement

This work has been supported by Iran Telecommunication Research Center (*ITRC*).

References

- [1] Barr M, A Reconfigurable Computing Primer, Miller Freeman Inc., 1998.
- [2] Betz V., VPR and T-VPack1 user's manual (Version 4.30), <http://www.eecg.toronto.edu/~vaughn>, 2000.
- [3] Bobda C., Synthesis of dataflow graphs for reconfigurable systems using temporal partitioning and temporal placement, Ph.D thesis, Faculty of Computer Science, Electrical Engineering and Mathematics, University of Paderborn, 2003.
- [4] Cong J., Ding Y., Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs, IEEE Transactions on CAD, 1994, pp.1-12.
- [5] Karthikeya M., P. Gajjala, B. Dinesh, Temporal partitioning and scheduling data flow graphs for reconfigurable computer, IEEE Transactions on Computers, vol. 48, no. 6, 1999, pp.579-590.
- [6] Kaul M., Vemuri R., Optimal temporal partitioning and synthesis for reconfigurable architectures, International Symposium on Field-Programmable Custom Computing Machines, 1998, pp.312-313.
- [7] Luk W., Shirazi N., P.Y.K. Cheung, Modeling and optimizing runtime reconfiguration systems, in: K.L. Pocek, J. Arnold (Eds.), Proceedings of IEEE Symposium on FPGA's Custom Computing Machines, IEEE Computer Society Press, 1996, pp. 167-176.
- [8] Maestre R., Kurdahi F., Fernandez M., Hermida R., Bagherzadeh N., Singh H., A framework for reconfigurable computing: task scheduling and context management, IEEE Transactions on VLSI Systems, vol. 9, no. 6, 2001, pp. 858-873.
- [9] Mehdipour F., Saheb Zamani M., Sedighi M., An Integrated Temporal Partitioning and Physical Design Framework for Static Compilation of Reconfigurable Computing System, Microprocessors and Microsystems, vol. 30, no. 1, Feb 2006, pp. 52-62.
- [10] Micheli G.D., Synthesis and optimization of digital circuits, McGraw-Hill, 1994.
- [11] Ouass I., Govindarajan S., Srinivasan V., Kaul M., Vemuri R., An Integrated Partitioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures, Proceedings of the Reconfigurable Architecture Workshop, 1998, pp. 31-36.
- [12] Sentovich E M, SIS: A system for sequential circuit analysis, Tech. Report No.UCB/ERLM92/41, University of California, Berkeley, 1992.
- [13] Shirazi N., Luk W., Cheung P.Y.K., Automating production of runtime reconfiguration designs, in: K.L. Pocek, J. Arnold (Eds.), Proceedings of IEEE Symposium on FPGA's Custom Computing Machines, IEEE Computer Society Press, 1998, pp. 147-156.
- [14] Spillane J., Owen H., Temporal partitioning for partially reconfigurable field programmable gate arrays, IPPS/SPDP Workshops, 1998, pp. 37-42.
- [15] Tanougast C., Berviller Y., Brunet P., Weber S., Rabah H., Temporal partitioning methodology optimizing FPGA resources for dynamically reconfigurable embedded real-time system, Microprocessors and Microsystems, vol. 27, 2003, pp. 115-130.
- [16] Zhang X., Ng K.W., A review of high-level synthesis for dynamically reconfigurable FPGA's, Microprocessors and Microsystems, vol. 24, 2000, pp.199-211.