Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2010, Article ID 709487, 5 pages
doi:10.1155/2010/709487

Research Article

Multi-FPGA Partitioning Method Based on Topological Levelization

Nabil Kerkiz, Amr Elchouemi, and Don Bouldin

Electrical & Computer Engineering, University of Tennessee at Knoxville, TN 37996, USA

Correspondence should be addressed to Nabil Kerkiz, kerkiz@yahoo.com

Received 29 April 2009; Revised 18 January 2010; Accepted 17 February 2010

Academic Editor: Ishfaq Ahmad

This paper presents a partitioning method based on topological ordering and levelization. The proposed method, termed RPL, performs multi-FPGA partitioning by taking into account six different partitioning constraints. We also compare RPL to two existing algorithms. The first approach is a hierarchical partitioning method based on topological ordering (HP). The second approach is a recursive algorithm based on the Fiduccia and Mattheyses bipartitioning heuristic (RP). Experimental results on seven application benchmarks mapped onto three different hardware architectures demonstrated that the proposed RPL approach achieved fewer partitions in less time when compared to the RP and HP algorithms.

1. Introduction

An adaptive computing system (ACS) composed of programmable logic components can serve as a flexible hardware accelerator for applications in domains such as image processing and digital signal processing. However, the mapping of applications onto ACSs using the traditional methods can take months for a hardware engineer to develop and debug. To enable application designers to automatically map their applications onto ACSs, a software design environment called CHAMPION was developed at the University of Tennessee. This environment permits high-level design entry using the Cantata graphical programming software [1, 2] from KRI. CHAMPION hides from the user the low-level details of the hardware architecture and the finer issues of application mapping onto the hardware.

An overview of the design flow of CHAMPION is shown in Figure 1. To map a Cantata graphical application onto ACS, the Cantata workspace format is first converted to a graph-based netlist. Data width matching is then performed to fix the mismatched data path where the bit width of the net carrying the data is larger or smaller than the bit width of the net receiving the data. The next step in the mapping process is to synchronize the netlist by determining the sizes and locations of the delay buffers necessary to balance the data traveling time of the various concurrent data paths. The algorithm proposed in [3] is employed to solve the data synchronization problem. Partitioning is then performed to subdivide the netlist into multiple subnetlists which will be implemented across the multi-FPGA system. After partitioning, the graph-based subnetlists are translated into structural VHDL files. The structural VHDL files for the subnetlists are merged with the VHDL files specifying the ACS architecture, internal interface logic for each programmable logic component, and global signals on the ACS. The merged VHDL files are synthesized using commercial synthesis software tools. Place and route software tools are then used to map the synthesized netlists to the programmable logic components and to obtain the programming bit files. The final step in the design flow is to automatically generate the host program which initializes the ACS board and downloads the programming bit files for each programmable logic component.

2. Hardware Architectures

Three different adaptive computing systems were chosen to implement the CHAMPION applications. These include the Wildforce-XL, the MSP, and the SLAAC boards. Since there are many resources available on the boards, it was decided to use a constrained configuration of the boards for

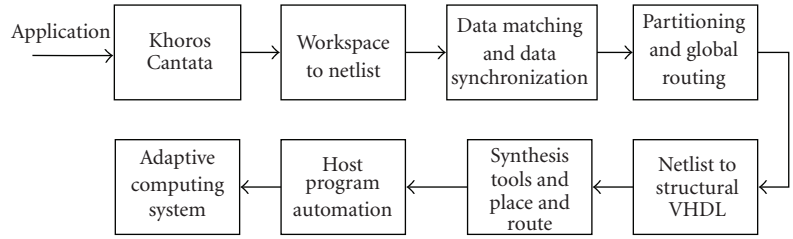

Figure 1: Overview of the design flow of CHAMPION.

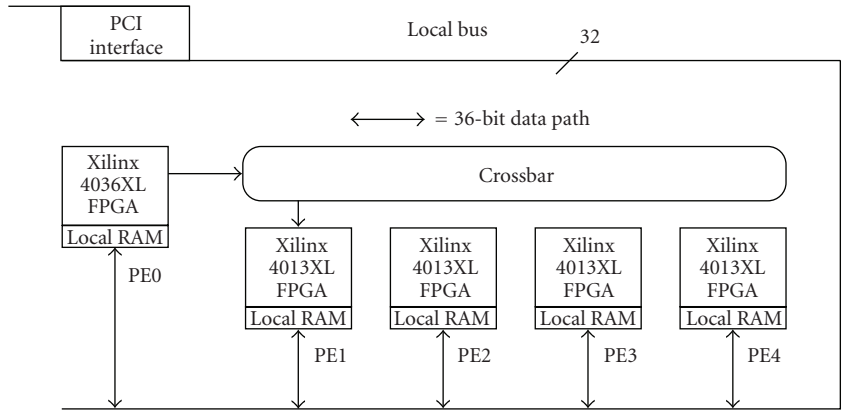

Figure 2: Wildforce-XL as used in the CHAMPION project.

the automatic implementations. This reduces the problem complexity to a more manageable level. A diagram showing the configuration of the Wildforce-XL board as used in this project is shown in Figure 2. All communication with the host is done through the SRAM associated with each processing element (PE). The crossbar is used only to provide a 36-bit path from CPE0 to PE1.

For the constrained implementation, it was also decided that the direction of all connections between processing elements would be fixed so that all signals would pass in one direction only. The board topology became a linear array, with all signals starting in CPE0 passing to PE1. No signals can run from PE1 back to CPE0. Similarly, all signals from PE1 run to PE2, with no signals allowed to pass back from PE2 to PE1. The MSP and the SLAAC boards are used in a similar manner where both boards have bigger resources compared to the Wildforce-XL.

3. Problem Formulation

A CHAMPION application is represented by a directed acyclic graph (DAG) $G = (V, E)$ where V is the set of nodes representing hardware glyphs, and E is the set of directed edges representing interconnections between glyphs. Nodes and edges in G are assigned positive values representing the area taken by glyphs and the net weights.

The partitioning problem consists in finding a partition of the initial netlist into minimal number of subnetlists $P = \{P_1, P_2, \ldots, P_k\}$ where each of these can be implemented on a single target FPGA. An FPGA is characterized by its size, I/O pins number, and local RAM. The partitioning problem is based on the following constraints: (1) capacity per partition, (2) number of I/O pins per partition, (3) each partition being able only to have a fixed number of RAM access modules, (4) input module and output module having to be placed in the first partition and in the last partition, (5) temporal partitioning constraint, (6) maintaining the acyclic constraint.

The first two constraints are used to meet the limitations of a single FPGA device. The third constraint deals with the memory access for each FPGA. The architecture of the ACS imposes that a fixed number of local RAMs are available to each FPGA for data writing and data reading. Therefore, a partition can only contain a certain number of RAM access modules. The fourth constraint deals with reading and writing the data. The input data are read first and supplied to the rest of the glyphs. The resulting data must be written to the hard disk via the output module. The fifth constraint deals with temporal partitioning of the ACS board. A single configuration of the board is the same as the configuration of all available FPGAs. If the entire application cannot fit in one board configuration, then multiple configurations of the board are necessary. When multiple configurations are used, storage of intermediate results between board configurations is needed. In this case, one RAM read hardware glyph must be added at the beginning of each configuration and one RAM write hardware glyph must be added at the end of each configuration. The sixth, and final, constraint requires maintaining the direction of the hypergraph so that all edges are pointing the same way.

4. Partitioning Approaches

In the past several years, many partitioning approaches and algorithms have been proposed to solve the FPGA partitioning problem [4–10]. One conventional partitioning approach for large netlists is to first apply clustering, topological ordering, and/or level construction schemes to reduce circuit complexity and then apply a set-covering method to reduce the number of required PEs. Several algorithms have been developed and used this approach to produce good results for specific applications [4, 5, 7–9]. Sanchis [6, 10] showed how to adapt a multiple-way network partitioning algorithm to work with two commonly used cost functions [6]. The method required reorganizing the way in which gain updates are computed with each cell move, in order to maintain relatively low time complexity. Kuznar and Brglez [4] introduced a new recursive partitioning paradigm which combines partitioning, replication, optimization, to be followed by another recursion of partitioning [4]. The algorithm demonstrated feasible partitions of some large netlists such that the number of device partitions is smaller than minimum lower bounds postulated initially. Wong et al. [8] presented a clustering-based acyclic multiway partitioning algorithm and showed that clustering can effectively be used to improve the overall partitioning results by including the acyclic property of the network [8]. The clustering step used a modified maximum fan-out free cone decomposition to perform clusters from a sparser network and the objective function was to minimize the cut cost between partitions. All of the above approaches perform partitioning on flattened netlists which do not consider a given hardware architecture with a large number of physical constraints. Temporal partitioning and its associated hardware resources is a very challenging task when an existing procedure is considered for development. Furthermore, the formation and property of a cluster or a level construction may be very sensitive to the complexity and quality of the partitioning procedure. Thus, it is important to choose an objective function that fits the target application. In this work, the relatively large number of constraints and the unique hardware architecture make it very difficult to adapt an existing algorithm without performing major surgery.

The previous discussion led us to consider a few existing methods for development and the introduction of a new partitioning approach. For solving the partitioning problem, three different approaches were investigated in this work. In the first and the second approaches, we discuss the development and implementation of two existing algorithms. The first approach is a hierarchical partitioning method based on topological ordering (HP). The second approach is a recursive algorithm based on the Fiduccia and Mattheyses bipartitioning heuristic (RP). We extend these algorithms to handle the RAM access constraint, the acyclic constraint, and the temporal partitioning constraint. We also introduce a new recursive partitioning method based on topological ordering and levelization (RPL). In addition to handling the partitioning constraints, the new approach efficiently addresses the problem of minimizing the number of FPGAs used and the amount of computation thereby overcoming the weaknesses of the HP and RP algorithms.

4.1. HP Algorithm. The algorithm, pointed out in [5], was modified and extended to handle the CHAMPION partitioning constraints. Given a topological sorting solution L of all nodes, we can partition the list L from left to right into K subnetlists $P = \{P_1, P_2, \ldots, P_k\}$ such that the constraints mentioned above are not violated. Initially all nodes are in the linear ordering L and the first block P_1 is empty. At each step, we select a node i from L and put it into P_1. The algorithm moves nodes into P_1 until the capacity constraint or the RAM access constraint of the partition is violated. Once one of these constraints is violated, the algorithm checks whether the interconnect constraint of the current PE is satisfied. If this interconnect constraint is violated, the algorithm rolls back the moves until the constraint is met. If there exists more than one candidate node for back rolling, the algorithm starts an optimization step to maximize the size of the current partition. The process is repeated by creating a new block P_2 and applying the same procedure to the remainder of L. The process stops when the list L becomes empty and all nodes are in P.

4.2. RP Algorithm. The second approach to this problem is a recursive algorithm based on the Fiduccia and Mattheyses (FM) bipartitioning heuristic. The partitioning approach, discussed in [4], was modified and extended to handle the CHAMPION netlist. In general, a random method is used to generate an initial bipartition for the FM heuristic. An arbitrary random bipartition can no longer be used here since it may violate the acyclic constraint. To create an initial solution, nodes are moved from the linear ordering array L into the current partition P_1 until the capacity constraint of P_1 has been met. In this process, each application of the bipartitioning procedure produces one feasible subnetlist and the remainder netlist. In the first iteration, the entire netlist R_0 is partitioned into one feasible solution, P_1, that meets the constraints of the first FPGA on the board, and the remainder partition, R_1. When the partitioner finds a feasible solution for the current partition P_1, the nodes in P_1 partition become locked. Subsequent iterations apply the same procedure to the remainder until all resulting partitions meet the constraints.

4.3. RPL Algorithm. In this section, we present our new approach for solving the partitioning problem. This algorithm applies existing ideas to graph partitioning; however, they have not been used in this manner previously. The algorithm strategy is based on two steps: the level construction step and the partitioning step. Given a topological sorting solution of all nodes, we can construct multilevels $L = \{L_1, L_2, \ldots, L_n\}$ of nodes N with a modified version of breadth-first search (BFS) such that nodes in level L_i appear before nodes in level L_j and the nodes in level L_j must be

Table 1: Partitioning results for the Wildforce-XL, the SLAAC, the MSP boards.

Netlists	#CLBs	#RAMs	#Nodes	#Nets	RPL algorithm		RP algorithm		HP algorithm	
					#Partitions	Time(s)	#Partitions	Time(s)	#Partitions	Time(s)
Partitioning results for the Wildforce-XL platform										
ATR	4885	14	101	234	20	2	23	9	Not feasible	—
R29	519	7	29	28	9	<1	Not feasible	—	Not feasible	—
R300	7845	11	301	311	21	3	23	26	25	4
R600	17640	9	702	714	23	21	24	138	29	34
R1000	24533	23	1005	1041	55	101	58	236	58	184
R1300	33120	25	1303	1243	64	225	66	501	72	424
R1500	41453	34	1502	1497	88	371	91	863	Not feasible	—
Partitioning results for the SLAAC-1V platform										
ATR	4885	14	101	234	4	<1	5	2	5	1
R29	519	7	29	28	2	<1	2	<1	2	<1
R300	7845	11	301	311	4	<1	4	11	4	2
R600	17640	9	702	714	3	11	3	89	3	19
R1000	24533	23	1005	1041	8	63	8	174	8	131
R1300	33120	25	1303	1243	8	127	8	379	8	289
R1500	41453	34	1502	1497	11	193	11	434	11	351
Partitioning results for the MSP platform										
ATR	4885	14	101	234	12	1	14	7	14	1
R29	519	7	29	28	6	<1	6	<1	6	<1
R300	7845	11	301	311	10	3	10	21	10	3
R600	17640	9	702	714	7	21	7	113	7	29
R1000	24533	23	1005	1041	21	76	21	222	22	172
R1300	33120	25	1303	1243	23	181	24	471	24	401
R1500	41453	34	1502	1497	32	273	33	604	34	503

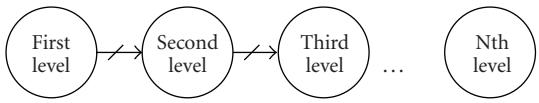

Figure 3: Reduced form after level construction.

successors of the nodes in level L_i. Afterwards, the resulting flow is reduced to a form shown in Figure 3.

Each level consists of a subset of nodes and no level can have more than one RAM access node. Level construction is a one-time step and is denoted as a preprocessing step. Given a levelization solution $L = \{L_1, L_2, \ldots, L_n\}$, we can partition L from left to right into K subnetlists $P = \{P_1, P_2, \ldots, P_k\}$. Initially all nodes are in L and the first block P_1 is empty. First, we start the process by moving n levels from L and put them into P_1 until P_1 has met the capacity constraint or the RAM access constraint is violated. At this moment, we mark the last level L_i moved to P_1 and its successor L_{i+1}. Then we start an optimization step by moving nodes across the marked levels. The optimization step is based on the benefit function discussed in [5]. Since only two levels are involved in the optimization step, the computation amount is reduced significantly. If the algorithm fails to find a valid solution, then we reduce the size of P_1 by removing the last level moved to P_1 and the optimization step is repeated. We repeat this process by creating a new block P_2 and applying the same procedure to the remainder of L. The process stops when L becomes empty and all nodes are in P. The pseudocode for the RPL algorithm is shown in Algorithm 1.

```
Input: G(N, E), D = devices;
Output: P1, P2, P3, ... , Pk
Create levels L = {L1, L2, ..., Ln};
k = 0; j = 1;
Proceed = True;
While (Proceed)
{
   Create new partition Pk;
   while (violation = False)
   {
    move level Lj to Pk; check constraints;
    mark the last level moved to Pk as Lj and its
    successor Lj+1; j = j + 1;
}
 Optimize Pk subject to the current device Dk;
 if (L = Dk+1) /* if the remained of L fit into
        the next device Dk+1 */
 Proceed = False; else k = k + 1;
}
```

Algorithm 1

5. Experimental Results and Discussion

The three partitioning approaches were implemented successfully in the C++ language. These algorithms were run on several netlists targeting different hardware architectures. Some of these netlists were generated randomly by using a random netlist generator. The automatic target recognition (ATR) application, which was implemented automatically from cantata workspace to the Wildforce platform, is a relatively complex netlist. The ATR was first implemented manually to assist in the development of function libraries and hardware for use in the CHAMPION system. The R29 netlist, which was generated manually, is a very challenging one utilizing a high number of RAM nodes. To the best of our knowledge, there exist no benchmarks that represent our partitioning constraints. For this reason, a random netlist generator to produce benchmarks that represent "real world" designs was developed in this research. The netlists R300-1500 were produced randomly with the random netlist generator.

Table 1 shows the partitioning results for the three hardware architectures. From the results, the following observations can be made. First, for several examples the RPL algorithm produced fewer partitions and faster timing compared to those produced by the RP and HP algorithms. The average run time was measured on a SUN workstation and it includes the time required to construct the levelization step for the RPL algorithm. For a specific configuration, the partitioning task is a one-time step and therefore the algorithms run time becomes important if the mapping process is repeated frequently.

This RPL algorithm improvement of achieving valid partitions and short average run times is due to the one-time level construction step used by the RPL algorithm. This was investigated by skipping the levelization step where the RPL algorithm produced similar results without significant improvements compared to those produced by the RP and HP algorithms. Since the RAM access constraint is a very challenging one, the preprocessing step was able to solve any conflicts associated with this constraint before starting the partitioning process. Second, the run time and the number of partitions produced varied with the selected hardware architecture. For the three partitioning algorithms, the run time and the number of partitions were reduced significantly by targeting hardware architectures with bigger I/O, bigger FPGA capacity, and bigger RAMs. Third, the partitioning results show that the HP and the RP approaches have difficulties in finding valid partitioning results for some of the netlists, which utilize a high number of RAM nodes, by targeting a hardware architecture with fewer resources. Referring to Table 1, the HP and RP algorithms failed to produce a feasible solution for the R29 when we targeted the Wildforce-XL. The problem arises when the RAM nodes are close to each other in the hypergraph netlist. The RAM access constraint is equivalent to the locking of a certain number of nodes in a netlist in which these nodes are prevented from moving freely across the cuts. This constraint is not affecting the movement of the particular node only but its neighbor nodes too. Another problem is the acyclic constraint. If one RAM node is locked in one partition, then the successors of this node are locked too because the movement of the successor nodes across a cut will violate the acyclic constraint.

6. Conclusion

In this paper, we have presented a partitioning method based on topological ordering and levelization (RPL). We also discussed the development and implementation of two existing algorithms. The first approach is a hierarchical partitioning method based on topological ordering (HP). The second approach is a recursive algorithm based on the Fiduccia and Mattheyses bipartitioning heuristic (RP). We extended these algorithms to handle the CHAMPION partitioning constraints by targeting different hardware architectures. Experimental results on a number of benchmarks demonstrated that the RPL algorithm achieved high performance when compared to the RP and HP algorithms.

Acknowledgment

The authors gratefully acknowledge the support of DARPA Grant no. F33615-97-C-1124.

References

[1] J. R. Rasure and C. S. Williams, "An integrated data flow visual language and software development environment," *Journal of Visual Languages and Computing*, vol. 2, no. 3, pp. 217–246, 1991.

[2] J. Rasure and S. Kubica, *The KHOROS Application*, Development Environment, Khoral Research Inc., Albuquerque, NM, USA, 2001.

[3] X. Hu, S. C. Bass, and R. G. Harber, "Minimizing the number of delay buffers in the synchronization of pipelined systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 12, pp. 1441–1449, 1994.

[4] R. Kuznar and F. Brglez, "PROP: a recursive paradigm for area-efficient and performance oriented partitioning of large FPGA netlists," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 644–649, November 1995.

[5] B. Stanley, *Hierarchical multiway partitioning strategy with hardware emulator architecture intelligence*, Ph.D. dissertation, Georgia Institute of Technology, 1997.

[6] L. A. Sanchis, "Multiple-way network partitioning," *IEEE Transactions on Computers*, vol. 38, no. 1, pp. 62–81, 1989.

[7] F. Vahid and D. D. Gajski, "Clustering for improved system-level functional partitioning," in *Proceedings of the 8th International Symposium on System Synthesis*, pp. 28–33, Cannes, France, September 1995.

[8] E. S. H. Wong, E. F. Y. Young, and W. K. Mak, "Clustering based acyclic multi-way partitioning," in *Proceedings of the 13th IEEE Great Lakes Symposium on VLSI*, pp. 203–206, 2003.

[9] W.-J. Fang and A. C.-H. Wu, "Multiway FPGA partitioning by fully exploiting design hierarchy," *ACM Transactions on Design Automation of Electronic Systems*, vol. 5, no. 1, pp. 34–50, 2000.

[10] L. A. Sanchis, "Multiple-way network partitioning with different cost functions," *IEEE Transactions on Computers*, vol. 42, no. 12, pp. 1500–1504, 1993.