Dynamic Placement Applications into Self Adaptive Network on FPGA

Petr Honzík
ČIP plus, s.r.o.
Milínská 130
Příbram, 26101, Czech Republic
Email: peters@cip.cz

Jiří Kadlec
Department of Signal Processing, UTIA AV CR v.v.i.
Pod vodárenskou věží 4
Praha 8, 182 08, Czech Republic
Email: kadlec@utia.cas.cz

Abstract— **The presented work deals with reconfigurable systems with Self Adaptivity based on the FPGA technology. The work is based on partial dynamic reconfiguration of FPGA devices and analyzes Self Adaptive systems, their elements and features. The main part introduces placement algorithms and Step Adaptive algorithm for improving mapping on running network. The tests of algorithm are done on the sets of the test applications.**

Keywords: FPGA, reconfiguration, adaptivity, placement, network-on-chip

I. Introduction

The era of a single computer per one person became on era of pervasive computing with many computers per one person. The pervasive computing era started with an expansion of embedded systems. The near future will increase the number of multiprocessors and heterogeneous hardware cores systems. These cores will have to process a huge amount of various data coming from the environment. Current cores can not fulfill requirements of the future multi-cores. The future multi-cores will work independently of any central controller, and can adapt to environment that will change during life time of system.

The future multi-cores will require much wider functionality with low power consumption and mainly reuse of the hardware resources. Today's computing systems have low hardware resource utilization, and the specific hardware pays for high performance by low utilization. This increases the cost of the hardware and power consumption. The future hardware has to have reconfiguration features that allow the change hardware functions of parts or the whole system without interrupting its run. This process will significantly increase the functionality and utilization of hardware

The current mainly centralized systems depend on central arbiters or controllers that synchronize and control the whole system. The future systems and their parts have to implement some level of self adaptivity and cooperation between parts of the system. Programmable logic array devices allow to implement most of the future requirements on today's devices. They cannot be compared with multiprocessors like IBM Cell and NVidia GPU with CUDA architecture but they can be used as an ideal starting platform that can prove methodology and features we find desirable in future systems.

This paper introduces an architecture based on autonomous computing units with self adaptive features connected by network-on-chip. The second section deals with a dynamic reconfiguration on FPGA device. The third section introduces a self adaptive feature. Then the fourth section presents static application placement into a network. The fifth section presents step adaptive algorithm for improving network placement. The sixth section present simulation test cases of placement and improving by Step-Adaptive algorithm. Finally, the seventh section concludes the paper.

II. Dynamic Reconfiguration

The FPGA devices allow fast prototyping of the hardware, and open fields for partial dynamic reconfiguration that can be used to change functions of parts of a system on chip. The current FPGA devices can contain several smaller microcontrollers with attached hardware accelerators that can act as a strong platform for future systems on chip with many of the features we described above [5].

It opens a new dimension in FPGA devices, increases variability and adaptability of the hardware. For this reason the functional density is defined and the basic condition for maximizing the function density is specified as reconfiguration time << execution time.

A real example of the combination of an ASIC device with an FPGA device is Atmel AT94K FPSLIC that has a hard core AVR type microcontroller connected to the programmable gate array. Another solution is the Xilinx Virtex device with the hard core Power PC processor attached to the programmable gates array. The future will use this model on a single chip to multi cores with many elements containing microcontroller with reprogrammable hardware accelerators. We used these two platforms as reference building block for our future research of the self adaptive network [1].

III. Self Adaptive Network

The Self-Adaptivity can be defined as the ability of a system to adapt to an environment. It is done by allowing components to monitor environment and change their behaviour in order to preserve or improve the operation of the system according to some defined criteria [2].

We will define Self-Adaptive Element (SAE) as architecture based on Self-Adaptive system. The SAE is a indivisible computing entity with local autonomous decision process occurs that affect its own operation. The following is structure of SAE:

- The computing engine processes data. It must provide the necessary processing power to handle future complex algorithms.
- The observer is responsible for monitoring the computation process and other runtime parameters.
- The controller is in charge of actually taking all decisions regarding the ongoing computation task.
- The communication interface is responsible for the management of Self-Adaptive assemblies.

We compared parameters of the NoC topologies to choose best NoC topology for SAN. The 2D-Mesh is the best topology that brings scalability, suitable communication cost and hardware cost doesn't exceed FPGA restrictions [3].

IV. Application Placement

We are looking for placement process of running applications on the network that will increase network performance and decrease communication cost.

We define network model as a connection direct graph G(P,L), where each vertex $p_i \in P$ represents a node and the edge $l_{ij} \in L$ represents a connection between two nodes. C introduces the cost of path between nodes p_i and p_j.

The application model can be written as a pair A(V,E). Each vertex $v_i \in V$ represents one task, and each edge e_{ij} represents an interaction between task v_i and task v_j.

We have placement algorithm that assigns tasks $v_i \in V$ to nodes $p_j \in P$ in the network. The placing process defined to places groups of tasks $v_i \in V$ to groups of nodes $p_j \in P$ according to constrains defined by the application area and the network topology.

$$V \mapsto P : \forall v_i \in V, \exists p_j \in P; place(v_i) = p_j$$

The placement algorithm finds the best position for task v_{best} in the network. The best position is determined by the distance from its predecessor tasks already placed on the network. The algorithm finds a node in a group of unused nodes only. The distance is counted from the distance matrix implement as memory array.

V. Self-Adaptive Placement

The running network accommodates and releases lots of applications during its life. The applications are accommodated irregularly according to incoming requests from the environment and release network when processes all data. These two operations cause fragmentation of tasks in the network. A fragmented network has higher power consumption and longer data processing.

We developed a Step-Adaptive algorithm that improves parameters of the network without interrupting its work. The algorithm improves the fragmentation of tasks. The algorithm is based on statistical information available at each node that contains a task and works autonomously and independently without a centralized unit.

The algorithm tries to adapt the current placement of a task on nodes by removing tasks independently step-by-step. The algorithm incrementally adapts task placement on the network according to the current communication load and distribution in the network communication channels.

The Step-Adaptive Algorithm looks for the best placement of all running applications on the network and use network cost C_{net} as parameters used for measuring the adaptive process of the network as a whole. Further we use the flit cost C_{flit} that says how expensive was to deliver flit from source to current task.

The main principle of the algorithm is to move a task from its current node to a neighbour node with the biggest port cost C_{port} of the current node. The task moves in the same direction as the data come from. If the task meets a task with the biggest port cost C_{port}, it tries to move around. If the task meets the source task, the algorithm stops.

Each node has an independent counter with a random seed that invokes the Step-Adaptive Algorithm for the node. The node needs to know only its port cost and occupation and port cost of its neighbour nodes. It allows to adapt a node independently of any central unit and central information.

The port cost calculates loading of each port by incoming flits that are processed in the node. It shows the direction of incoming data [4].

$$C_{port(t+1)} = C_{port(t)} + \alpha(\beta_{magnetic} C_{flit(t)} - C_{port(t)})$$

α is a positive step size parameter in the range $0 \leqslant \alpha \leqslant 1$. βmagnetic increases the port cost of the node when the source task is a direct neighbour node of the current node. Each port of the node has its own port cost C_{port}.

A. *Moving rules*

The Step-Adaptive Algorithm moves tasks according to basic moving rules. Moving rules define behaving of tasks during adaptation. The rules are classified in two classes according to the number of incoming streams of flits to a task. We have rules for one-stream tasks and for multi-stream tasks. The rules are in Figure 1. Rules A, B, and C are valid for both one-stream and multi-stream tasks. Rules D and E are valid only for one-stream tasks and rules F and G are valid for multi-stream tasks.

VI. Simulation of the SAN

We simulated runs on the network with six sets of application that covers various types of real network using.

The sets covers stream type, parallel and mixed type of application with various communication loads.

The sets S1, S2 and S3, applications in set have different communication loads. The sets S4, S5 and S6 have same communication load in the set.

We developed two types of port cost measuring that have different hardware cost and have different communication load accuracy. Hop cost method counts only distance from source task and hardware cost is low. Path cost method combines distance from source task and communication load in each node that flit pass on its path. This method is more accuracy but hardware cost is higher.

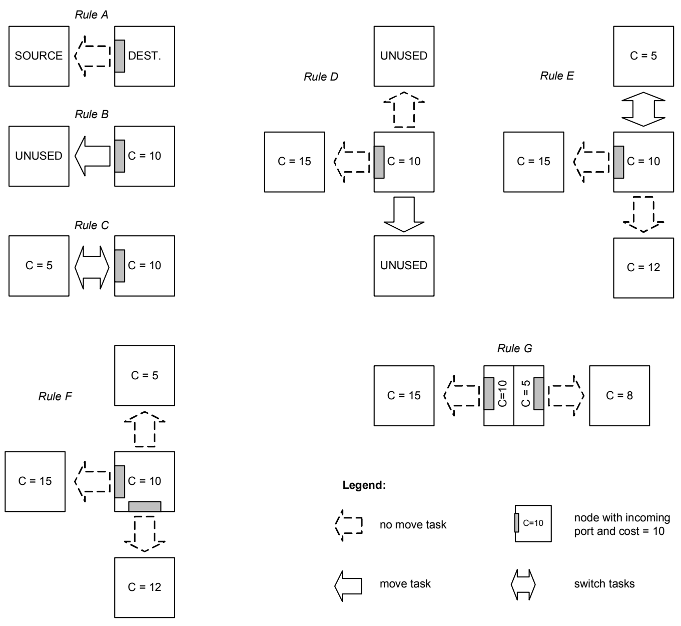

Figure 1. Rules for moving tasks across nodes.

We did 150 runs for each set of applications to get the behaviour of the Step-Adaptive Algorithm for both cost method. Each 25 runs from a total of 150 runs have a different magnetic coefficient to observe its impact to the Step-Adaptive Algorithm. The application set can be split to two groups. First, application sets have different communication loads, S1, S2 and S3. Second, S4, S5 and S6 application sets have the same communication load.

To compare the simulation results we count the average network cost for each test set and for each magnetic coefficient. The average cost is not count over all iterations of the improvement process, but after the network cost stabilized.

From the comparisons of both the methods and from the comparison graph in Figure 2 we can conclude that for applications with different communication loads we can use both the methods with similar results. For applications with the same communication loads we need the magnetic coefficient and the path method can have better results for parallel applications.

The standard deviation for all the test sets and both methods show that the Step-Adaptive Algorithm has stable results for all test sets.

A. Stream Application Tests

Stream type applications are most used in image processing and computer graphics. Parallelism of these applications can be seen on more stream applications that work concurrently in one network. From this reason we want to know the behavior of the Step-Adaptive Algorithm in the network with only stream applications. As two test cases we chose five stream applications, each with four nodes, and two stream applications, each with nine nodes. All the applications have the same rate of incoming flits. We want to see the interaction of the applications as the placement, and their effectiveness in terms of use of the network resources us.

We use two different starts of the network, random placement and placement presented above to shows that both reached the minimal network cost. The run with the placement above reached the minimal network cost two times faster than with the random placement. The final result gets improved by 37%. In the case of the random placement the improvement is 79.6%, but because the starting network cost is much worse.

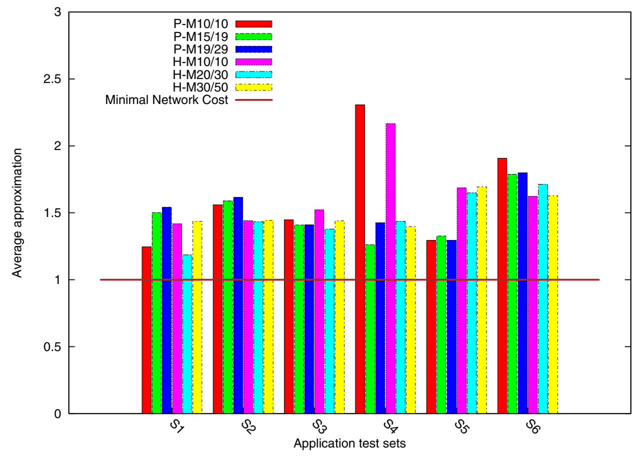

Figure 2. Bar graph compares average approximation of application test sets. Explanation: P means path cost method; H means hop cost method; Mpp/hh means size of βmagnetic parametr. S1, S4: stream type applications; S2, S5: paralel type applications; S3, S6 mixed type applications. S1, S2, S3: different communication loads; S4, S5, S6: same communication loads.

The simulation results for application with same communication load shows that behaviour of applications during the improving process depends on the magnetic coefficient. When the magnetic coefficient is smaller than 30, the applications oscillate, and the network cost can significantly increase. When the magnetic coefficient is higher than 30, the magneticity between the neighbor nodes is big enough to generate successful result, see Figure 3.

We can say that all applications in both the tests were reshaped to chains that are near to most efficient in terms of communication. The conclusion of the tests for stream applications is that the improvement process leads a near optimal placement. From the results of the sets with same communication load we can see that, a higher magnetic coefficient produces better results with the Step-Adaptive Algorithm.

B. Placement and Step-Adaptive Algorithm

We test the Step-Adaptive Algorithm in real network life when new applications are placed to the network and finished applications are released from the network. For placing a new application we use the placement algorithm presented above and the random placement algorithm that places an application to a running network.

In real network case an application with a heavy communication load is placed to free nodes in the network. But these nodes cannot be really the most suitable nodes, and the application can significantly increase the network cost. A typical case is when we place several low communication load applications, and later we place heavy communication load applications.

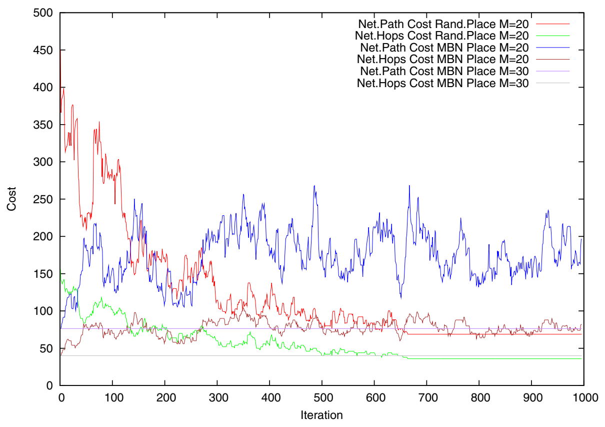

Figure 3. Improving network cost Cnet. Explanation: Random and Multi-Best node placement algorithm improved by Step-Adaptive algorithm. M means βmagnetic parametr. Measured by hop and path cost method.

The heavy communication load application cannot be placed near the data source because of the previous application. It will increase mainly the network path cost. The Step-Adaptive Algorithm starts moving tasks with a heavy communication load to their source and push low communication load tasks out from their nodes.

When we compare the types of applications running in the network, we can see that the stream applications reached better improvement by the Step-Adaptive Algorithm than the parallel applications. It is due to heavy transfer between nodes with serial connections.

Another conclusion is that more small applications have better improvements than fewer bigger applications. It is caused by the independency of the small applications. The tasks of the small applications can be moved easier than large applications. Large applications create long walls from tasks tied together by their communications. From the test of real application cases in the network we can conclude that the placement algorithm can bring some improvement compared to the random placement, but there is still enough space to improve the network cost. This improvement can be done by the Step-Adaptive Algorithm. The Step-Adaptive Algorithm brings improvements in most of the cases from 20% to 40% compared to the network cost gains by the presented placement algorithm.

VII. Conclusion

We introduced the simulation framework with the Step-Adaptive Algorithm and two methods for driving the improvement process in the running network. We introduced two methods for driving the Step-Adaptive Algorithm.

We designed the Step-Adaptive Algorithm for improving task placement in the network. With this we introduced the magnetic coefficient that holds neighbor tasks together.

From the simulation we can see that the magnetic coefficient can significantly influence stream type applications with the same communication load. Generally the magnetic coefficient can control all stream type applications placing. We can use the magnetic coefficient to force an application to group tasks in a chain and decrease delays in an application.

In the case of parallel applications the magnetic coefficient doesn't work because the task works with more input streams. The Step-Adaptive Algorithm is designed to improve placement of applications in a running network. Injecting and releasing an application from the network increase task fragmentation in the network, and we need the Step-Adaptive Algorithm for task defragmentation.

From the simulation of a real life time of the network we can see that the Step-Adaptive Algorithm can significantly improve the placement.

We simulated a Self-Adaptive network in the simulation framework to find suitable network topology and improvement algorithm. All restrictions for simulation come from the FPGA platform that is our target platform for the Self-Adaptive network. They require us to find an undemanding topology and algorithms that can be implemented on the FPGA devices. From this reason we prefer the Step-Adaptive Algorithm with the hop method for implementation. The result of the hop method is similar as the path method, but the implementation of the cost calculation hardware block is much easier than for the path method.

Acknowledgment

This work was supported by the SMECY project; project number: Artemis JU 100230, MŠMT 7H10003.

References

[1] Bartosinski, R., Danek, M., Honzık, P. and Matousek, R., Dynamic reconfiguration in FPGA-based SoC designs, in Proceedings of the 8th IEEE DDECS 2005, Sopron, Hungary pp. 129–136.

[2] Danek, M., Philippe, J.-M., Honzik, P., Gamrat, C. and Bartosinski, R., Selfadaptive networked entities for building pervasive computing architectures, in 'ICES 2008: Berlin, Heidelberg, pp. 94–105.

[3] Salminen, E., Kulmala, A. and Hamalainen, T., On network-on-chip comparison, in 'Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on', pp. 503 –510.

[4] Sutton, S. R. and Barto, G. A. (1998), Reinforcement Learning: An Introduction , MIT Press. ISBN 0262193981.

[5] Edson L. Horta, John W. Lockwood, David E. Taylor, David Parlour, "Dynamic Hardware Plugins in an FPGA with Partial Run-time Reconfiguration," Design Automation Conference, p. 343, 39th Design Automation Conference (DAC'02), 2002