Hosseinabady, M., & Nunez-Yanez, J. L. (2009). Run-time resource management in fault-tolerant network on reconfigurable chips. In International Conference on Field Programmable Logic and Applications, 2009 (FPL 2009), Prague. (pp. 574 - 577). Institute of Electrical and Electronics Engineers (IEEE). 10.1109/FPL.2009.5272400

RUN-TIME RESOURCE MANAGEMENT IN FAULT-TOLERANT NETWORK ON RECONFIGURABLE CHIPS

Mohammad Hosseinabady
m.hosseinabady@bristol.ac.uk
University of Bristol, Bristol, UK

Jose L. Nunez-Yanez
j.l.nunez-yanez@bristol.ac.uk
University of Bristol, Bristol, UK

ABSTRACT

This paper investigates the challenges of run-time resource management in future coarse-grained network-on-reconfigurable-chips (NoRCs). Run-time reconfiguration is a key feature expected in future processing systems which must support multiple applications whose processing requirements are not known at design time. This paper investigates a stochastic routing algorithm in a NoC-based system with dynamically reconfigurable tiles, able to cope with the dynamic behaviour of run-time task mapping. Experimental results show the efficiency of the proposed stochastic task mapping.

1. INTRODUCTION

Future multimedia consumer electronic products (e.g., mobile phones, set-top boxes, home theatre systems, etc.) will need to support very demanding applications like image, data and sound processing, cryptography, wireless communication, and streaming high-definition television. This wide range of applications and standards imposes new demands in chip design such as high integration, more reliability, high bandwidth required among functional modules, and reconfigurability. Run-time reconfiguration techniques provide the ability to change parts of the hardware to adapt to the requirements of an application. Using partial reconfiguration allows more tasks to be run in hardware and therefore reduces the application execution time compared with software running on an embedded processor. Task mapping and dynamic resource management are main issues in run-time reconfiguration platforms which are addressed in this paper.

1.1. Research Motivation and Contribution

The run-time reconfigurable platform that we focus on in this paper consists of a group of tiles communicating through a network of routers (Figure 1). A tile in the platform consists of two parts: static part and run-time reconfigurable (RTR) part. We assume that the static part contains a bus, a local memory, and a network interface (NI). We assume that the RTR part is a fine-grained reconfigurable architecture based on sea-of-logic-cells as is the case in current FPGA technology. The RTR part, which is connected to the bus, can host a hardware core at run-time. The hardware core can be a general purpose processor or a dedicated hardware with size not greater than the reconfigurable region area. This platform will run an application represented by a hardware/software partitioned and scheduled task graph.

Reconfiguration overhead and *dynamic task requests* are the two issues in the task mapping on a run-time reconfigurable architecture that have motivated this paper. The hardware reconfiguration overhead is a considerable overhead in an RTR hardware based on current FPGA technology which cannot be ignored, for example in the Xilinx Virtex family, and it can be up to a few milliseconds. In addition, because of the dynamic behaviour of applications, the requested task in the SoC is not uniform over time. This introduces randomness in the location and time of the active task, which in turn makes difficult to locate a task in the platform.

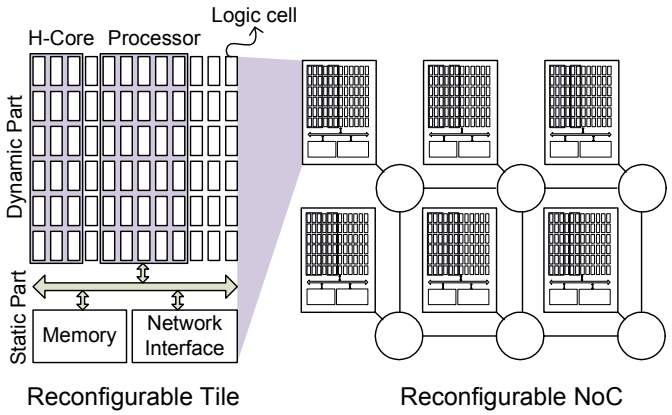

Figure 1 Run-time reconfigurable NoC under investigation

In this paper, to map a dynamically requested task graph, we propose a stochastic routing algorithm to find a task or to find a proper reconfigurable region for mapping the task. Reliability and network topology independence are the two benefits of a stochastic routing algorithm. The stochastic routing algorithm randomly chooses an outgoing port among the non-faulty ports at each router. Therefore, if there exists a fault detection mechanism in each router, then the stochastic routing simply ignores the faulty ports.

1.2. Related Work

Broadcast or multicast routing algorithms are among the stochastic algorithms presented in the literature [1][2]. Our stochastic method reserves the data path between source and destination tiles during task mapping using a single request flit and this way differs from flooding technique. This is done to avoid a lot of resource reservation which results in a huge overhead on the mapping algorithm. Nollet et al. [3] investigate a run-time task assignment heuristic in a multiprocessor SoC containing fine-grain reconfigurable hardware tiles. While they have considered fully reconfigurable FPGAs in a heterogeneous multi-processor

The authors acknowledge with gratitude the support obtained from the EPSRC UK under grant number EP/E062164/1

SoC, in this work we consider the partial run-time reconfigurable feature of FPGAs. Chou et al. [4] propose a run-time mapping algorithm for run-time applications which are dynamically mapped onto a NoC-based multiprocessor SoC with different voltage levels. They propose incremental mapping method to address the energy and performance in real time. They use a global manager for system resource management which runs the mapping algorithm. Using a centralised global manager limits the scalability of this method for large NoCs. In addition, any fault in the controller makes the entire system fail.

The rest of this paper is organised as follows. The next section explains the proposed SoC architecture and the stochastic routing algorithm. Experimental results are presented in Section 3. Finally, the paper concludes with Section 4.

2. PROPOSED PLATFORM AND METHODOLOGIES

Our methodology gets a scheduled task graph and a mesh of reconfigurable tiles as its inputs. The proposed methodology incrementally maps the requested tasks of the task graph onto the tiles. Figure 2(a) shows a simple task graph with three tasks (i.e., t_1, t_2 *and* t_3). Let's assume task t_1 is mapped on tile *10,* and it needs task t_2. For this purpose, it sends a task request flit (TRF), which has information about task t_2 as well as some fields for routing algorithm, to its router. The router sends the TRF randomly to one of its neighbouring routers (i.e., router *11*). This router, which receives the TRF, sends the TRF to its attached tile to check that it can host (or has) the task t_2. The router registers itself to the TRF as a volunteer if it has the capability of hosting the task. Routers along the path reserve resources needed for the future data transmission and decrement the life-time field in the TRF. When the life-time of the TRF expires, the last router in the path sends back the TRF. The return TRF releases all unnecessary reserved path and activates the best tile along the path to run the task. It is difficult to find a suitable tile to host a task only by using a simple random walk routing. A good distribution of different PR region can improve the success of the random routing. In addition, some factors and metrics about the dynamic behaviour of the mapped task and PR regions are saved in tables in tiles which can be used to guide TRF towards a proper tile. These metrics are explained in Subsection 2.2.

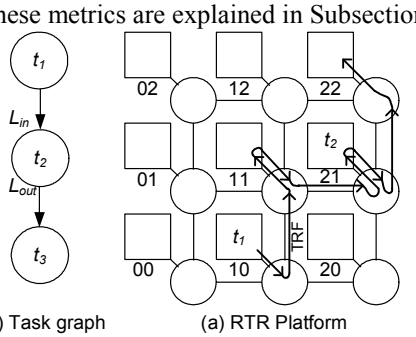

Figure 2 Simple idea of the proposed random routing

2.1. Topology

The number and type of resources and their distributions are important to successfully find a suitable resource during a stochastic task mapping scheme.

Allocating different types of partially reconfigurable regions is the first step to design a NoC topology. PRRs with different types can be PRR with different sizes (which we assume in this paper), different supply voltages, or different frequencies. We spread different resource types on the NoC considering the distribution of task types in the task graphs.

2.2. Task mapping guidelines

The random walk should follow the following constraints to be able to map a task to a PRR, efficiently.

Constraint 1: Finding an existing task
Constraint 2: Finding a PRR which fits the task
Constraint 3: Minimize the communication overhead between two communicating tasks

We define a few metrics to numerically denote these constraints. Network interface of tiles save these numbers as guidelines for the randomly travelled flit.

PRR factors: PR number (*PRN*) *and* PR distance (*PRD*) factors, which are computed for each port of a router for each PR type, denote the number and average distance of free PR regions of a specific type in the half-plane area on the side of the port, respectively. The randomly routing scheme uses these factors to tend flit towards the area with more available required PR regions. When a PR region hosts a task, the corresponding NI sends its new state to the neighbouring routers to inform those routers of the new state. Also, the tiles that update their states send the new states to their neighbours. So, after a while all NIs are able to update their tables.

Example: Let's consider the tile (2,3) of Figure 4(c). All PR region of types *a*, *b*, *c,* and *d* on the right hand side of this tile (i.e., $i > 2$) are 4, 4, 1, and 1, respectively. Therefore,

$PRN^{a}_{Right}(2,3) = 4, PRN^{b}_{Right}(2,3) = 4$

$PRN^{c}_{Right}(2,3) = 1, PRN^{d}_{Right}(2,3) = 1$

The average distance of the PRs of type *a* on the right hand side of this point is $PRD^{a}_{Right} = \frac{\sum_{i>2,\alpha(i,j)=a} t(i,j)}{\sum_{i>2,\alpha(i,j)=a} 1} = \frac{16}{4} = 4.$

Using the same scheme, we can compute *PRN*s and *PRD*s for the other three directions which are left, up, and down.

Task factors: Task number *(TN) and* Task distance (*TD*) factors, which are saved in NI and are computed for each port of a router for each task, denote the number and average distance of free mapped tasks in the half-plane area on the side of the port, respectively. Randomly routing scheme uses these factors to tend flit towards the already free mapped task in order to reduce task reconfiguration overhead. When a mapped task in a tile finishes its functionality, it sends a flit to its neighbouring tiles and informs them of the available free task in the tile which is a hop away. The neighbouring tiles update the distance to the task in a table and send a flit to inform their neighbours of the free task in a tile which is two hops away and other tiles do the same. These factors are computed in the same method of computing PRR factors.

Cost function (*CF*): while the four factors *PRN*, *PRD*, *TN*, and *TD* try to guide a flit towards the proper direction. The cost function, *CF*, evaluates different tile opportunities along the path. So, the algorithm can select the best tile along the traversed path to map the task. Considering the reconfiguration overhead and the incoming/outgoing bandwidth needed for the required task, the following equation computes the CF at each NI for the given task.

$$CF(t_j) = Recon_j + L_{in} * D(t_i, t_j) + L_{out} * \sum_{i\in\{L,R,U,D\}} PRD_i^k/4$$

The first term shows the reconfiguration overhead of each task. The second term computes the incoming communication bandwidth overhead in which *Lin* denotes the incoming bandwidth and $D(t_i, t_j)$ shows the number of hops between predecessor mapped task t_i and requested task t_j. The third term predicts the communication bandwidth overhead for the successor task of the requested task t_j in which L_{out} is the outgoing bandwidth and $\sum_{i\in\{L,R,U,D\}} PRD_i^k/4$ predicts the average distance to the available PRR for the successor task t_k.

2.3. Task mapping algorithm

In this subsection, we explain the router and NI algorithms with more details.

Router: An application is described by a scheduled task graph. In the proposed algorithm a predecessor task sends a task request flit (TRF) to find or map the successor task(s). The task request flit format for one task is shown in Figure 3. It contains *routing flags*, *task information*, and *life time* fields. Router which generates a random number less than *RANDMAX* use the routing flags to select the outgoing port. Routing flags computed by NI are three numbers which divide the range between 0 and *RANDMAX* into four intervals.

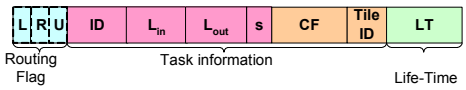

Figure 3 Task request flit format

Network interface: NI has two responsibilities for the routing algorithm: table updating and TRF updating.

Table updating: during the platform start-up, the *PRN*s/*PRD*s and *TN*s/*TD*s factors can be initialised in the tables. If we assume that there is no preloaded or hardwired task then all *TN*s/*TD*s factors are zero. When a tile hosts a task, it informs other tiles by sending an *updating flit* using a simple broadcasting algorithm. Note that this broadcasting method can be done whenever system is in steady state and there is enough resource for that. Because of the stochastic task mapping, we can still map tasks if one of the tiles cannot broadcast its new state. A NI which receives an updating flit (e.g., denotes adding/releasing of a task t_1 of type *a* at distance *d)* modifies its factors as follows:

Adding/Using a task:

$$PRD_i^a(n+1) = (PRD_i^a(n) * PRN_i^a(n) - d)/(PRN_i^a(n) - 1)$$
$$PRN_i^a(n+1) = PRN_i^a(n) - 1$$

Using/Replacing a task:

$$TD_i^t(n+1) = (TD_i^t(n) * TN_i^t(n) - d)/(TN_i^t(n) - 1)$$
$$TN_i^t(n+1) = TN_i^t(n) - 1$$

Releasing a task:

$$PRD_i^a(n+1) = (PRD_i^a(n) * PRN_i^a(n) + d)/(PRN_i^a(n) + 1)$$
$$PRN_i^a(n+1) = PRN_i^a(n) + 1$$
$$TD_i^t(n+1) = (TD_i^t(n) * TN_i^t(n) + d)/(TN_i^t(n) + 1)$$
$$TN_i^t(n+1) = TN_i^t(n) + 1$$

which *i*, *a*, *t* are indices to denote the direction (i.e., right, left, up, down), PRR type, and task identification, respectively.

Updating TRF: combining task factors we compute e_i^t factor as follows:

$$e_i^t = (TN_i^t/TD_i^t)/\sum_k (TN_k^t/TD_k^t);\ k,i \in \{L,R,U,D\}\ TD_k^t \neq 0$$

Also, combining PRR factors we compute f_i^a factor as follows.

$$f_i^a = (PRN_i^a/PRD_i^a)/\sum_k (PRN_k^a/PRD_k^a);\ k,i \in \{L,R,U,D\}\ PRD_k^a \neq 0$$

Then combining these two factors, e_i^t and f_i^a, we can calculate the routing flag for the requested task as below:

$$L^t = (e_L^t + f_L^a)/\sum_k (e_k^t + f_k^a) * MAXRAND$$

$$R^t = L^t + ((e_R^t + f_R^a)/\sum_k (e_k^t + f_k^a) * MAXRAND$$

$$U^t = L^t + R^t + ((e_U^t + f_U^a)/\sum_k (e_k^t + f_k^a) * MAXRAND$$

3. EXPERIMENTAL RESULTS

To evaluate the proposed technique, we simulate 10000 times the two applications shown in Figure 4(a), (b). The *App1* and *App2* application task graphs are based on the task graphs of a 263 decoder mp3 decoder (*App1*) [5], a multi-window display (MWD) [6] (*App2*).

We have considered four types of PR regions (PR regions with different sizes denoted by *a*, *b*, *c*, and *d*) which are shown on each task node in task graphs. The mapping of the resource graph on a 5x5 mesh topology is shown in Figure 4(c). The resource type distribution in this mesh topology is the same as resource type distribution in the two task graphs. As it can be seen, the resources available in 5x5 mesh topology are not enough to run two applications, simultaneously. Therefore, for running multiple applications we use a mesh of 10x10 nodes with the PR region type distribution the same as resource type distribution in task graphs. In the sequel, using these applications and mesh topologies, we evaluate the proposed methodology for mapping the tasks to the 10x10 mesh.

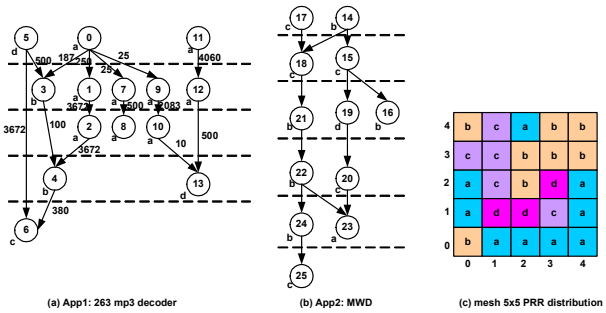

Figure 4 Three task graphs, their resource graph and PRR distribution on NoC

To investigate the behavior of the proposed methods in a loaded NoC, we mapped six applications (two 263mp3dec,

three MWD, and one 263mp3dec, consequently) on the 10x10 NoC. Figure 5(a) and (b) show the results of mapping these applications using the proposed stochastic method and random walk, respectively. Based on Figure 5(a), the proposed algorithm can find the proper tiles with high probability even in a heavy loaded NoC wheras the random walk needs much higher life time for request packets to be able to mapp an application in a heavy loaded NoC.

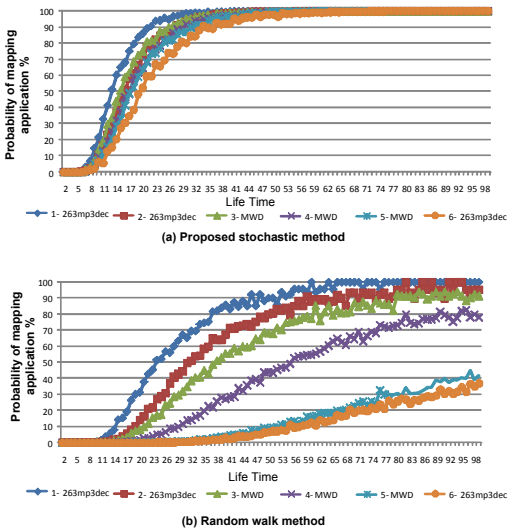

Figure 5 Probability of mapping applications with different size on 10x10 mesh

To handle the reconfiguration overhead issue the proposed method tries to reuse the already reconfigured tasks in tiles. To evaluate the task reuse feature of the proposed methodology, starting from tile (0, 0), we have mapped MWD application twice, consecutively, on the 5x5 and 10x10 mesh topologies for three different scenarios: simple random walk, the proposed method with only PRR factors, and the proposed method with PRR and task factors. Figure 6(a) and (b) shows these results for 5x5 and 10x10 mesh, respectively. As it can be seen, PRR factors can significantly improve the task reuse with compared to the simple random walk. In addition, using task factors also improves the task reuse, which is one of the paper contributions. As it can be seen, with the increase in mesh size from 5x5 to 10x10 the reusability decreases significantly. Regional based task mapping is a solution to handle this drawback. Because even if we can find an existing task which is in a distance from its predecessor task in the task graph, the communication overhead may be higher than the reconfiguration overhead. Investigating this technique will be part of our future work.

4. CONCLUSIONS

This paper has proposed a stochastic mapping and routing algorithm to run an application described by a task graph on a NoC-based reconfigurable platform. The paper has defined two kinds of factors to represent the information about the available reconfigurable resource and free mapped tasks in the tiles. These factors guide the stochastic routing to find suitable reconfigurable region and already mapped tasks in order to optimise reconfiguration overhead and find a feasible solution. The experimental results show the success of the defined factors in optimally map applications onto 5x5 and 10x10 mesh topologies compared to the simple random walk mapping scheme.

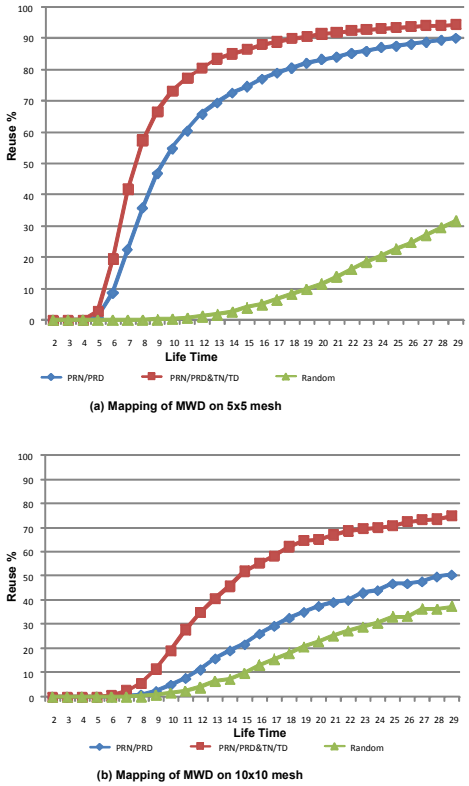

Figure 6 Percentage of task reuse when we map MWD twice consecutively with different TRF life time

REFERENCES

[1] T. Dumitras, and R. Marculescu, "On-Chip Stochastic Communication," in *Proceedings of Design, Automation and Test in Europe* (DATE'03), 2003.

[2] R. Karp et. al. Randomized rumor spreading. In *Proc. IEEE Symp. On Foundations of Comp. Sci.*, 2000.

[3] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal, "Run-time management of a MPSoC containing FPGA fabric tiles," *IEEE Transactions on Very Large Scale Integration* (VLSI) Systems, vol. 16, no. 1, 2008.

[4] C. L. Chou, U. Y. Ogras, and R. Marculescu, "Energy- and performance-aware incremental mapping for network on chip with multiple voltage levels," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, October 2008.

[5] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear-programming-based techniques for synthesis of network-on-chip architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 4, pp. 407-420, 2006.

[6] K. Srinivasan, and K. S. Chatha, "A low complexity heuristic for design of custom network-on-chip architectures," in *Proceedings of the conference on Design, automation and test in Europe* (DATE'06), pp.130-135, 2006.