

Run time Mapping of Adaptive Applications onto Homogeneous NoC-based Reconfigurable Architectures

Stefan Wildermann and Jürgen Teich

University of Erlangen-Nuremberg, Germany

{stefan.wildermann, juergen.teich}@cs.fau.de

Abstract—Tile-based NoC architectures have emerged as promising field-programmable hardware architectures which provide arrays of programmable processing units to support run time adaptation. However, efficient real-time support is required for executing adaptive applications which offline design approaches may not provide. In this paper, running dynamic master/slave applications on homogeneous NoC architectures is investigated. Such applications have the characteristic that their program structure may be adapted at run time by inserting or removing tasks to react to changing requirements that are not known a priori. A heuristic is presented for an energy- and performance-aware assignment of dynamically created tasks to the processing units. The provided experimental results give evidence of the benefits of the proposed methods for synthetic test-cases as well as an adaptive image processing application.

I. INTRODUCTION

The application area of embedded systems has considerably grown in recent years, and with that their complexities. The more complex a system grows, the more it has to support situations which are not predictable at design time as well as handle system or user requirements that change during operation. To achieve this, adaptive applications adjust their behavior at run time, e.g., initiated by a controlling system component or based on new user input or training data.

The dominant factors for embedded systems tailored for executing such applications are flexibility, computational efficiency, low power consumption, and a scalable communication infrastructure. Tile-based architectures, such as coarse-grained reconfigurable architectures and Multi-Processor Systems-on-Chip (MPSoCs), have emerged as promising reconfigurable platforms fulfilling these requirements. Reconfigurability is provided by programmable tiles which are distributed on the chip. This allows to change the application behavior by programming individual tiles at run time. The parallelism of applications can be exploited to satisfy the computational demands. A problem coming with this parallelization is, however, that new approaches for on-chip communication are required that scale with the architecture. Networks-on-chip (NoCs) have emerged as a paradigm for designing on-chip communication [1], [2].

Our paper aims at solving the problem of mapping adaptive applications which change their behavior and structure at run time while adapting to new requirements. Applications are considered where a central task (*master*) delegates work to processing tasks (*slaves*) which can run in parallel; a model very important in such fields as networking, multimedia, and signal processing. Adaptation of the program structure happens by spawning or removing slave tasks. Adaptation at run time may have various reasons:

- Signal filtering has to be adapted to changing signal characteristics.
- The application has time-variant resource constraints or data dependencies.
- The workload of an application increases or decreases.
- Functions are switched on or off by dynamic temperature management or power saving mode.
- System reliability has to be enhanced by executing redundant task instances.

The paper is organized as follows. Section II gives an overview of related work. The preliminaries of this work are provided in Section III. Section IV presents the heuristic for mapping dynamically created tasks. Experiments with synthetic test-cases as well with an adaptive image processing case study are given in Section V. Section VI concludes this paper.

II. RELATED WORK

A concept where reconfigurable resources are dynamically occupied and released is the worker helper model proposed in [3]. Worker components perform specific applications and sometimes require helper components to perform some services which therefore provide reconfigurable hardware. Comparable to the master/slave application model, resources of the helper components can dynamically be occupied by hardware modules and be released again after having finished. The work, however, does not take into account characteristics of the on-chip communication of NoCs. Run time mapping algorithms for NoC architectures are investigated in some works: [4] proposes a run time mapping algorithm that allows to map applications that arrive incrementally. The algorithm selects near convex regions on the NoC to map the application onto. However once mapped, the application must not change during its execution. A decentralized approach for application mapping is presented in [5] with relatively low processing times. In a homogenous system however, this implies a overhead for the PEs since they must be able to run parts of the operating system. [6] analyzes the *best-fit* and the *worst-fit* mapping heuristics as centralized strategies where the former generates a task distribution based on load balancing, and the latter generates a load concentrated to some units. On the one hand, load concentration may help reducing energy consumption due to communication, on the other hand, load balancing minimizes deadline misses. [7] proposes several run time mapping heuristics for dynamic applications with changing behaviors during execution. Interestingly, the simple nearest-neighbor (NN) algorithm generates efficient results.

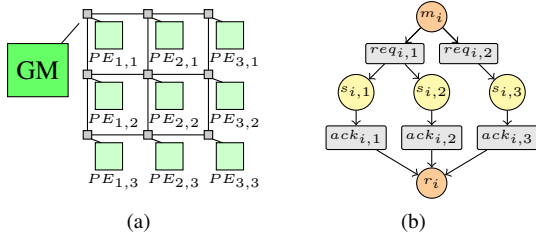


Fig. 1. (a) NoC architecture and (b) application graph of master/slave application.

III. PRELIMINARIES

A. NoC Architecture

In this paper, we focus on homogeneous NoC architectures composed of a 2D interconnect of *processing elements* (PEs), as shown in Fig. 1(a). It is defined as follows:

Definition 1: A homogeneous NoC is a $w \times h$ mesh consisting of PEs $PE_{x,y}$, $1 \leq x \leq w$, $1 \leq y \leq h$. Each PE is composed of a *processing unit* (PU) and a *router*. $R(PE_{x,y})$ gives the resources (e.g., memory) provided by the processing unit which may be consumed by tasks assigned to it.

A global manager (GM) is considered as a further component running the operating system and serving as a gateway for external system components which may use the platform as an hardware accelerator for their dynamic applications.

B. Master/Slave Application

To better explain the application characteristics, we will first formally define static master/slave applications, before detailing on dynamic applications.

Definition 2: A *master/slave application* i is defined as a bipartite directed graph $G(V_i, E_i)$, denoted as *application graph*, with vertices $V_i = P_i \cup C_i$ and edges E_i . The (soft) deadline of the application is given as d_i . An example of an application graph is shown in Fig. 1(b).

- The *processing tasks* $P_i = \{m_i, r_i\} \cup S_i$ consist of the master and the slave tasks. The master is composed of task m_i , responsible for distributing data to the slaves, and task r_i , for receiving results and acknowledgments from the slaves, respectively. The set S_i contains the slave tasks $s_{i,j} \in S_i$.
- *Communication nodes* C_i are divided into *request messages* $req_{i,k}$ and *acknowledgment messages* $ack_{i,j}$. Each request is sent by the master and may be received by several slaves to support multi-cast communication. After finishing its processing, each slave task sends the results or an acknowledgment back to the master. Edges E_i represent the data dependencies indicating sender and receivers of messages.

For each processing task $p \in P_i$, $pt(p)$ is the task's *processing time* and $rr(p)$ gives the *resource requirements* that are needed on a PE to ensure a successful execution of the task. Each communication node $c \in C_i$ has the property $ms(c)$ denoting the *message size* (bits).

Dynamic applications change their behavior at run time. In a master/slave application, this means that new slave tasks may be spawned, or existing ones may be released at time step t , initiated via the GM. Such an application is defined as a function over t :

Definition 3: A *dynamic master/slave application* i is defined as application graph

$$G(V_i(t), E_i(t))$$

where the set of slave tasks can vary over time and, with that, the communication: $V_i(t) = \{m_i, r_i\} \cup S_i(t) \cup C_i(t)$.

C. Task Mapping

Task mapping is defined as a function $\beta_t(p)$ that assigns each task p to a specific PE of the NoC for all applications in each time step t . When running n master/slave applications with associated application graphs $G(V_i(t), E_i(t))$, the mapping function is given as

$$\beta_t(p) = PE_{x,y}, \quad p \in \bigcup_{i=1}^n P_i(t). \quad (1)$$

Since each master, composed of tasks m_i and r_i , is seen as one entity, we have $\beta_t(m_i) = \beta_t(r_i)$ ¹.

Communication is assumed to be implemented by standard online NoC routing schemes. We therefore do not have to calculate a proper route offline, thus reducing the computational requirements of the online mapping approach. The problem of run time task mapping can now be formulated as follows:

Definition 4 (Run time Task Mapping Problem):

Given n applications and a mapping $\beta_{t-1}(p)$ for all $p \in P_1(t-1) \cup \dots \cup P_n(t-1)$.

Find mapping $\beta_t(p')$ for newly arriving task p' with following objectives:

- 1) Application performance is increased by placing nodes distributed on the PEs.
- 2) Minimize communication latency and communication energy consumption, both proportional to number of hops, according the approximation of [8].
- 3) Allow mapping of further incoming nodes to the system with minimal inter-processor communication overhead.

IV. RUN TIME TASK MAPPING

The proposed heuristic for solving the problem of Def. 4 is based on cost minimization. The function used for evaluating the costs for mapping a new slave task $s_{i,j}$ onto $PE_{x,y}$ at step t is given as

$$\begin{aligned} cost_t(s_{i,j}, PE_{x,y}) = & \alpha_1 \cdot W_t(s_{i,j}, PE_{x,y}) + \\ & \alpha_2 \cdot D_t(s_{i,j}, PE_{x,y}) + \\ & \alpha_3 \cdot N_t(s_{i,j}, PE_{x,y}) \end{aligned} \quad (2)$$

This function is a weighted sum with W_t approximating the *worst-case start time* of $s_{i,j}$ when being processed on $PE_{x,y}$, D_t considering a *communication metric*, and N_t is a *neighborhood metric*. Note that, if $PE_{x,y}$ has not sufficient resources left for processing $s_{i,j}$, the costs are set to ∞ .

The approximation of the worst-case start time corresponds to when all other tasks already mapped onto $PE_{x,y}$ are scheduled before $s_{i,j}$. Therefore, it is computed by accumulating the processing times $pt(p)$ of all tasks $p \in \mathcal{T}(PE_{x,y}, \beta_{t-1})$, i.e., all tasks being already mapped onto $PE_{x,y}$. The sum

¹From now on, we do not distinguish between m_i and r_i when considering the task mapping.

is normalized by dividing it through the deadline d_i of the application the new task $s_{i,j}$ belongs to:

$$W_t(s_{i,j}, PE_{x,y}) = \frac{\sum_{p \in \mathcal{T}(PE_{x,y}, \beta_{t-1})} pt(p)}{d_i}. \quad (3)$$

Note that we do not take into account the time for transmitting the messages. The transmission time would depend on the distance $PE_{x,y}$ and $\beta_{t-1}(m_i)$ as well as the actual message sizes. Both entities are taken into account when calculating the communication metric.

The purpose of the communication metric is to consider the communication energy consumption and the message transmission times for on-chip communication. Both are proportional to the number of hops between sender and receiver as well as the message sizes in bits. Let the communication nodes $req_{i,k}$ and $ack_{i,j}$ be the request and acknowledgment message of $s_{i,j}$, respectively. In case $req_{i,k}$ is a multi-cast message, transmitting it has only to be considered if there is no other receiver already mapped onto $PE_{x,y}$. The size that is actually to be transmitted is set to:

$$size_{i,j} = \begin{cases} ms(ack_{i,j}), & \text{if } req_{i,k} \in \mathcal{C}_i(PE_{x,y}, \beta_{t-1}) \\ ms(req_{i,k}) + ms(ack_{i,j}), & \text{else} \end{cases} \quad (4)$$

where $\mathcal{C}_i(PE_{x,y}, \beta_{t-1})$ denotes the set of communication nodes that are exchanged between the slaves $s \in \mathcal{T}(PE_{x,y}, \beta_t) \cap S_i$ and the master. The communication metric is then calculated as

$$D_t(s_{i,j}, PE_{x,y}) = \frac{hops(PE_{x,y}, \beta_{t-1}(m_i)) \cdot size_{i,j}}{max_dist(\beta_{t-1}(m_i)) \cdot ms_{max} \cdot 2}. \quad (5)$$

It is normalized using ms_{max} , the size of the largest message mapped on the NoC, and $max_dist(\beta_{t-1}(m_i))$, the maximal distance of $\beta_{t-1}(m_i)$ to the corners of the NoC.

Finally, the purpose of the neighborhood metric is to consider the third objective formulated in Def. 4 by evaluating the neighborhood $\mathcal{N}(PE_{x,y})$ of the current target $PE_{x,y}$. The idea is to rate a PE depending on the amount of resources in its vicinity that are occupied by tasks of the same application. At the same time, it takes into account the amount of resources occupied by tasks of other applications. This metric is inspired by rules known from *Cellular Automata*, where cells change their occupation depending on the states of their neighbors. In our current implementation, $\mathcal{N}(PE_{x,y})$ is composed of the element itself as well as the elements to the North, South, West, and East. The neighborhood metric is calculated as the rate between the overall amount of occupied resources $occ(PE_{x,y})$ and the amount of resources occupied by the application itself $occ_i(PE_{x,y})$ with

$$occ(PE_{x,y}) = \sum_{\substack{p \in \mathcal{T}(PE, \beta_{t-1}) \\ PE \in \mathcal{N}(PE_{x,y})}} rr(p) \quad (6)$$

$$occ_i(PE_{x,y}) = \sum_{\substack{p \in \mathcal{T}(PE, \beta_{t-1}) \cap P_i(t) \\ PE \in \mathcal{N}(PE_{x,y})}} rr(p). \quad (7)$$

Now, the proposed metric for target $PE_{x,y}$ is calculated as:

$$N_t(s_{i,j}, PE_{x,y}) = \begin{cases} 1 - \frac{occ_i(PE_{x,y})}{occ(PE_{x,y})}, & \text{if } occ(PE_{x,y}) > 0 \\ 1, & \text{else} \end{cases}. \quad (8)$$

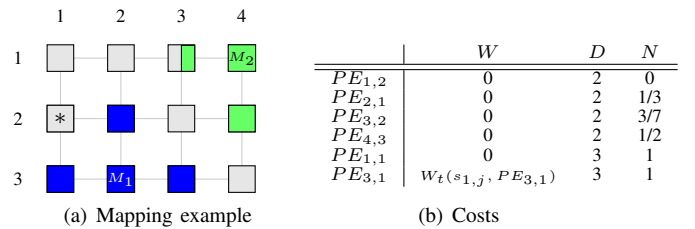


Fig. 2. Example for calculating the cost function. (a) shows a mapping by displaying the resources occupied by two applications. (b) lists the costs of all PEs with free resources for mapping a new slave task $s_{1,j}$. The minimal costs arise for $PE_{1,2}$ (*).

In Fig. 2, an example which illustrates the influence of this metric is given. Fig. 2(a) colors the resources occupied by two applications in a NoC. 50% of the resources of $PE_{3,1}$ are consumed, and 100% of all remaining PEs which are occupied. The values of the proposed cost function for all PEs with available resources are listed in Fig. 2(b). Here, the minimal costs arise for $PE_{1,2}$. The neighborhood metric enforces to map a new slave task of application 1 further away from application 2, leaving sufficient resources for both applications to grow efficiently in the future. The metric also contributes to form *near convex regions* for which it has been shown in [4] that applications may be mapped more efficiently.

V. EXPERIMENTAL RESULTS

Our simulation framework consists of two parts: a software tool and a NoC simulator. The software tool simulates the application behaviors, which may dynamically spawn and release slave tasks, and the GM, implementing the proposed mapping algorithm. When the mapping changes, the NoC simulation is started which provides the run-time properties of the current system configuration. We use the publicly available tool NIRGAM [9] for a cycle accurate SystemC simulation of the on-chip communication, extended by a simulation model for the PUs. We tested the proposed mapping heuristics with four different settings:

- The proposed mapping heuristic (RT+N): $\alpha_1 = 0.45$, $\alpha_2 = 0.45$, and $\alpha_3 = 0.1$.
- The mapping heuristic without the neighborhood metric (RT): $\alpha_1 = 0.5$, $\alpha_2 = 0.5$, and $\alpha_3 = 0$.
- Communication-aware mapping, comparable to NN in [7] or worst-fit in [6]: $\alpha_1 = \alpha_3 = 0$, $\alpha_2 = 1$.
- Performance-aware mapping based on load distribution (LD): $\alpha_1 = 1$, $\alpha_2 = \alpha_3 = 0$.

A. Results for Random Test-cases

We generated two classes of random test-cases. Each consisted of 16 applications. The masters are uniformly distributed on a 9×9 NoC with a clock of 150 MHz. Each application is assigned with parameter p_{grow} that specifies the probability that a new slave task is created. In class A, the slaves' resource requirements were randomly chosen between 10% and 25% of the PU resources, request message sizes between 1 and 5 Bytes, and acknowledgments between 1 and 2 Bytes. Parameter p_{grow} ranged from 40% to 50%. For class B, the resource requirements were between 25% and 75% of the PUs resources. Growing probabilities ranged from 40% to 80%.

The results are depicted in Fig. 3. Shown are the overall communication and the average relative processing time. The

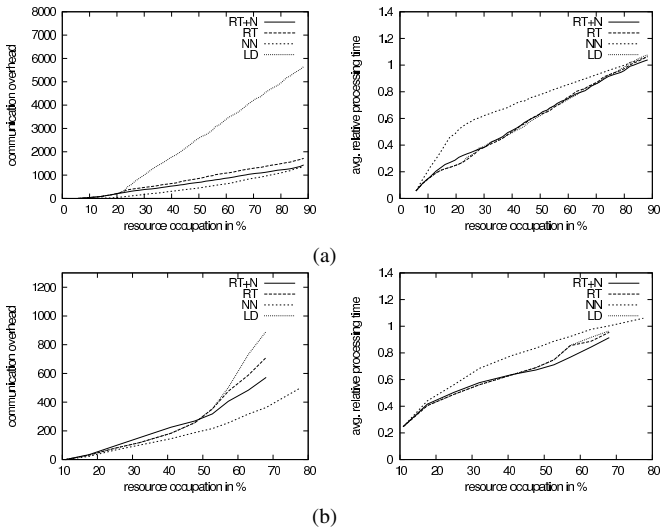


Fig. 3. Results for communication overhead and average relative processing times for random test-cases (a) class A and (b) class B with growing resource requirements.

communication overhead is the sum of the sizes of transmitted messages times the number of hops. The relative processing time of an application is its latency divided by its deadline. The communication overhead, as indicator for the systems energy consumption, grows very fast for LD but does not lead to a better performance compared to RT+N and RT. The communication-aware NN strategy maps tasks onto the closest PE with sufficient resources. Since parallelism is not exploited, the performance degrades very fast. The neighborhood metric of RT+N actually helps to reduce communication. Especially for the very dynamic test class B, RT+N even allows to significantly reduce the performance degeneration, as indicated by the average relative processing time. This means that, if a low on-chip communication is the key requirement, NN is the preferred strategy. In all other cases, RT+N should be used.

B. Results for Adaptive Image Processing

The proposed heuristic was also tested for an adaptive image processing application. The classifier from [10] is used for *face detection*. The basic idea of this classifier is to divide the image window into so-called *receptive fields*: small, partly overlapping sub-images. Each of these sub-images is processed by a neural network. Their results are combined to decide if the overall image window contains a face or not. The main contribution of this classifier is that it can be adapted at run time by inserting or removing neurons of the neural networks. The classifier in our experiments used 12 neural networks which were implemented as separate master/slave applications and uniformly distributed on the NoC. Each master receives the image data of its receptive field via the GM. It forwards the data to its neurons which are implemented as slave tasks. They calculate the neuron activities and return their results to the master. Finally, each master calculates the weighted sum of these results which is then sent back to the image processing application via the GM. In our experiments, the classifier was adapted at run time to training images by adding and removing neurons according to the learning algorithm of [10]. We tested the case-study for a 7×7 NoC clocked at 50 MHz and consisting of PUs with 1024 Bytes memory. The simulation results of the mappings during training are

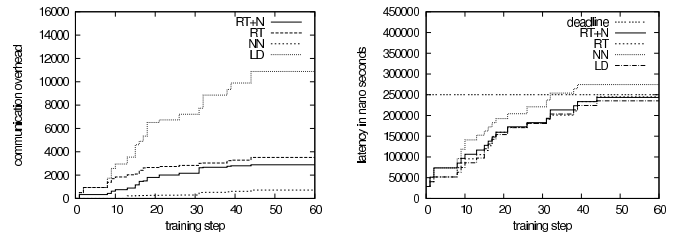


Fig. 4. Results for communication overhead and latency for case study.

shown in Fig. 4. Again, NN produces solutions with low communication overhead since parallelism is not exploited. However, NN exceeds the predefined deadline at training step 32. Mappings with all other strategies stay below the deadline. RT+N performs best when regarding both, the energy and performance objectives.

VI. CONCLUSION

In this paper, mapping of adaptive applications onto tile-based reconfigurable systems using Networks-on-chip communication schemes was investigated. The master/slave application model was proposed to define such applications where adaptation of the program structure happens by inserting or removing slave tasks. A novel task mapping algorithm was presented for an energy- and performance-aware assignment of dynamically created tasks to the processing units. This heuristic includes a neighborhood metric inspired by rules known from *Cellular Automata* which allows to decrease the communication overhead produced by dynamic applications. The experimental results have shown that this metric is the preferred strategy when considering the conflicting objectives performance and energy consumption for task mapping.

REFERENCES

- [1] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan 2002.
- [2] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *DAC '01: Proceedings of the 38th conference on Design automation*. New York, NY, USA: ACM, 2001, pp. 684–689.
- [3] D. Merkle, M. Middendorf, and A. Scheidler, "Self-organized task allocation for computing systems with reconfigurable components," *Parallel and Distributed Processing Symposium, International*, vol. 0, p. 271, 2006.
- [4] C.-L. Chou, U. Ogras, and R. Marculescu, "Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 10, pp. 1866–1879, Oct. 2008.
- [5] M. Al Faruque, R. Krist, and J. Henkel, "ADAM: Run-time agent-based distributed application mapping for on-chip communication," *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pp. 760–765, June 2008.
- [6] E. W. Briao, D. Barcelos, and F. R. Wagner, "Dynamic task allocation strategies in MPSoC for soft real-time applications," *Design, Automation and Test in Europe, 2008. DATE '08*, pp. 1386–1389, March 2008.
- [7] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs," in *RSP '07: Proceedings of the 18th IEEE/FIP International Workshop on Rapid System Prototyping*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 34–40.
- [8] T. T. Ye, G. D. Micheli, and L. Benini, "Analysis of power consumption on switch fabrics in network routers," in *DAC '02: Proceedings of the 39th annual Design Automation Conference*. New York, NY, USA: ACM, 2002, pp. 524–529.
- [9] NIRGAM homepage, <http://nirgam.ecs.soton.ac.uk/>.
- [10] S. Wildermann and J. Teich, "A sequential learning resource allocation network for image processing applications," in *Proceedings of the 8th International Conference on Hybrid Intelligent Systems*, Barcelona, Spain, Sep. 2008, pp. 132–137.