Undisrupted Quality-of-Service during Reconfiguration of Multiple Applications in Networks on Chip

Andreas Hansson[1], Martijn Coenen[2] and Kees Goossens[2]
[1]Eindhoven University of Technology, Eindhoven, The Netherlands
[2]Research, NXP Semiconductors, Eindhoven, The Netherlands
m.a.hansson@tue.nl, martijn.coenen@nxp.com, kees.goossens@nxp.com

Abstract

Networks on Chip (NoC) have emerged as the design paradigm for scalable System on Chip (SoC) communication infrastructure. Due to convergence, a growing number of applications are integrated on the same chip. When combined, these applications result in use-cases with different communication requirements. The NoC is configured per use-case and traditionally all running applications are disrupted during use-case transitions, even those continuing operation.

In this paper we present a model that enables partial reconfiguration of NoCs and a mapping algorithm that uses the model to map multiple applications onto a NoC with undisrupted Quality-of-Service during reconfiguration. The performance of the methodology is verified by comparison with existing solutions for several SoC designs. We apply the algorithm to a mobile phone SoC with telecom, multimedia and gaming applications, reducing NoC area by more than 17% and power consumption by 50% compared to a state-of-the-art approach.

1 Introduction

Systems on Chip (SoC) grow in complexity with an increasing number of processors, memories and accelerators integrated on a single chip. These heterogeneous high-complexity chips are programmable and integrate a rich set of applications [5, 20], e.g. PDA phones with mp3 players, cameras, radios and gaming.

The individual *applications* are combined into *use-cases*. Figure 1 shows six use-cases, u_0 through u_5, of a mobile phone SoC. As seen in the figure, applications typically span multiple use-cases, e.g. *roam* continues as the foreground applications, *mp3 play* and *mpeg play*, change. The different use-cases have different traffic patterns and Quality-of-Service (QoS) requirements, e.g. bandwidth and latency constraints that the communication infrastructure must efficiently accommodate.

Networks on Chip (NoC) have emerged as the design paradigm for scalable on-chip communication architectures, providing better structure and modularity [1, 3, 8, 21]. By offering QoS guarantees to individual *flows* between cores, NoCs decouple computation from communication [19]. Decoupling enables independent design and validation of every part of the SoC by ensuring that real-time requirements are met independent of other parts of the system [8]. Application guarantees are, however, traditionally only given *within* a use-case and do not cover use-case transitions.

Existing approaches to multi-application NoC design [15, 16] cannot give QoS guarantees to applications across use-cases as the binding of flows to network resources is done per use-case. Even if the application requires the same service, e.g. *phone* in u_0 and u_1, the resources used to provide the requested service are potentially different before and after reconfiguration.

As the label *global reconfiguration* in Figure 1 illustrates, traditionally a transition from use-case u_i to u_{i+1} involves: 1) tearing down the flows of all applications in u_i, and then subsequently 2) instantiating all flows in u_{i+1} by programming the NoC with a new *configuration* that decides the binding of flows to buffers, paths and time-slots. Also the applications spanning both u_i and u_{i+1} thus have their flows closed and later re-opened. Not only does this cause a disruption in delivered services, but it leads to unpredictable reconfiguration times as in-flight transactions must be allowed to finish when tearing down flows between cores [12, 17].

In many domains, e.g. consumer electronics, medical IT and automotive, there are inherent demands for seamless use-case switches for certain applications. The only

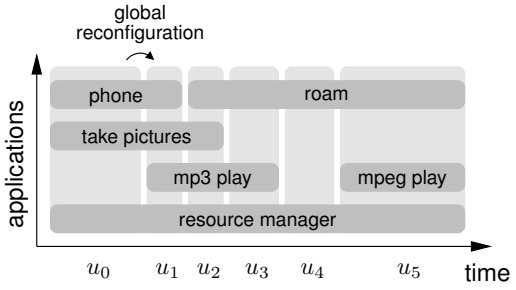
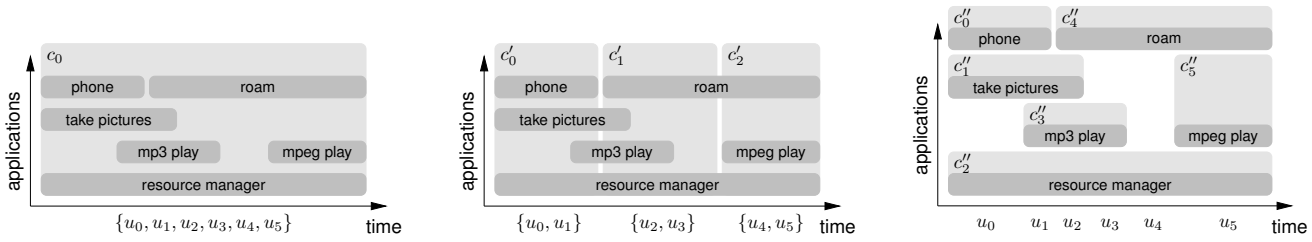


Figure 1. Multi use-case SoC design.

978-3-9810801-2-4/DATE07 © 2007 EDAA



(a) Worst-case (b) Compound modes (c) Partial reconfiguration

Figure 2. Comparison of reconfiguration models.

way to provide this using traditional approaches to NoC design is to avoid reconfiguration altogether. This is achieved by mapping and configuring for a synthetic worst-case use-case covering the requirements of all use-cases [15]. While delivering application guarantees, requirements are over-specified with a costly NoC design as the result [16]. Moreover, if the architecture of the NoC is given, it may not be possible to find a configuration that meets the worst-case use-case requirements, even though configurations exist for each use-case individually.

In this work we address the problem of mapping multiple QoS-constrained applications onto a NoC that provides resource virtualisation (e.g. connections). We present: 1) a theoretical framework that applies to any such NoC, and 2) a mapping algorithm that exploits the partial reconfiguration capabilities of our NoC. This enables QoS guarantees for applications *across* use-cases without using the synthetic worst-case configuration, reducing NoC area by more than 17% and power consumption by 50% for a mobile phone SoC with telecom, multimedia and gaming applications.

The remainder of the paper is structured as follows. We start by introducing related work in Section 2. Next, the problem domain is described in Section 3 and formalised in Section 4. The algorithm, which solves the NoC mapping and configuration problem under multiple application constraints, is described in Section 5. Experimental results are shown in Section 6. Finally, we conclude in Section 7.

2 Related work

Methodologies for dynamic *run-time reconfiguration* of NoCs are presented in [17, 22]. Both works assume little or no design time knowledge about the applications and defer mapping decisions to run time. While offering maximal flexibility, guaranteeing that an application can be given the requested QoS or even be instantiated at all is difficult due to possible resource fragmentation over time. Mitigating the problem necessitates complex migration schemes with large unpredictable delays [17].

Much work is focused on complete NoC design flows [2, 8] and the problem of mapping cores onto NoC architectures for a single pre-defined use-case [9, 11, 14, 18, 23]. All works are limited to a single set of communication constraints, obtained either from a single use-case or from a single trace containing multiple use-cases.

NoC design for multiple use-cases is addressed in [15] by generating a synthetic worst-case use-case. The result is one NoC configuration spanning all potential use-cases, illustrated by c_0 in Figure 2(a). In [16], the lack of scalability in the synthetic worst-case solution is addressed by introducing aggregated use-cases, called *compound modes*. As seen in Figure 2(b), configuration is done per compound mode. Undisrupted QoS is provided within the constituent use-cases, but not during reconfiguration.

Existing design approaches distinguish only one axis of configuration granularity, namely time, at which the *whole* NoC is reconfigured. The temporal granularity ranges from a single use-case [9, 14] to a synthetic worst-case [15] with compound modes in between [16]. In this work we introduce a second axis (the vertical axis in Figure 2) distinguishing between different *spatial* granularities. While the aforementioned related works reconfigure globally (time only), this work considers reservations on a per-application basis (time plus space).

This work is, to the best of our knowledge, the first to provide application QoS guarantees during NoC reconfiguration. The model we propose differs from existing research in that it assigns (virtual) resources to applications rather than to use-cases. As we shall see, this model together with our mapping algorithm enables partial reconfiguration of the NoC and undisrupted QoS at a low cost.

3 Problem description

We assume static applications, defined at design time, with non-migratory tasks already mapped onto cores, such as processors and accelerators. The bandwidth and latency constraints of the communication flows are determined beforehand by static analysis or simulation. The use-cases (combinations of applications) are assumed to be given together with a specification of which applications are *persistent*, i.e. that require undisrupted QoS during use-case transitions.

Our goal is to derive: 1) a core port to network interface (NI) port mapping (if not already specified by the designer), and 2) a number of pre-computed compositional NoC configurations that fulfil the QoS requirements of the applications. These configurations bind the logical flows to physical *buffers* in the NIs, to *paths* through the router network, and to *time slots* on the network links.

By deriving configurations per application, the different applications are associated with their own virtual resources, multiplexed in time and space, and can be removed or added independently of one another. Consider the schematic example in Figure 3 where the resources used by different applications are shaded in different colours[1]. If the *phone* application is stopped and replaced by *roam*, then the other running applications remain unaffected. Not only does this enable undisrupted QoS, it is also beneficial from a scalability point of view as the reconfiguration operation and coordination can be distributed, e.g. a reconfiguration processor (CPU) per subsystem.

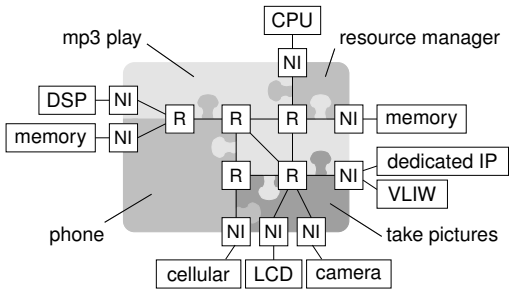


Figure 3. Network virtualisation.

Configurations are created and verified at *design time*. Hence, *run time* choices are confined to choosing from the set of fixed configurations. While limiting the run-time choices to a set of predefined use-cases, this is key as it enables us to guarantee, at *compile time*, that all application constraints are satisfied once a configuration is instantiated.

The computed configurations are stored in for example off-chip memory. A run-time configuration manager then instantiates these configurations as a result of trigger events [17]. The actual protocol for reconfiguration is outside the scope of this paper.

Note that the suggested methodology does not exclude the possibility of using unclaimed residual resources to instantiate additional applications at run time. These applications are however limited to best-effort services.

4 Problem formulation

Inter-core communication is specified on the level of applications, characterised as graphs.

Definition 1. From the perspective of the NoC, an application a is a directed multigraph, $a(P_a, F_a)$, where the vertices P_a represent the *core ports*, and the arcs F_a represent the set of *flows* between the ports. Each flow in the application $f \in F_a$ is associated with a minimum throughput constraint, $t(f)$, and a maximum latency constraint, $l(f)$. Source and destination of f are denoted $s(f)$ and $d(f)$.

[1]The channels and buffers are time-multiplexed between flows but only the spatial division of resources is shown.

The set of applications is denoted A. We define the complete set of core ports P as the union over all applications, $P = \bigcup_{a \in A} P_a$. Similarly, the complete set of flows $F = \bigcup_{a \in A} F_a$.

A *use-case* $u \subseteq A$ is defined by the applications that run simultaneously, e.g. $u_0 = \{$*phone*, *take pictures*, *resource manager*$\}$ in Figure 1. The set of use-cases $U \subseteq \mathcal{P}(A)$ grows exponentially with the number of applications if there are no limitations on which applications can be combined. However, some applications, such as *phone* and *roam* are inherently mutually exclusive. Others, like *mp3 play* and *mpeg play*, might be infeasible to instantiate simultaneously because they use the same resources. An undirected graph, as shown in Figure 4, expresses which applications may run in parallel. Every fully connected subgraph corresponds to a use-case, with six of them shown in Figure 2. Let $U_a \subseteq U$ denote the set of use-cases containing an application a.

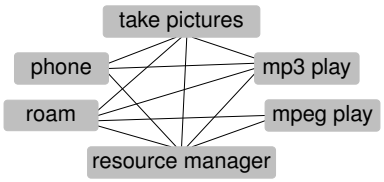


Figure 4. Use-case specification.

NoCs are represented by interconnection network graphs.

Definition 2. An *interconnection network graph* g is a strongly connected directed multigraph, $g(N, E)$. The set of vertices N is composed of two mutually exclusive subsets, N_R and N_{NI} containing *routers* (R) and *network interfaces* (NI) respectively.

The set of edges E represent the physical network channels. More than a single physical channel is allowed to connect a given pair of routers, but an NI is always connected to a single router through one egress and one ingress channel. Source and destination of e are denoted $s(e)$ and $d(e)$.

A path $\pi \in \operatorname{seq} E$ from source $n_s \in N_{NI}$ to destination $n_d \in N_{NI}$ is a sequence of channels in the set of all possible paths $\Pi(n_s, n_d)$. Slot tables in the NIs govern allocation of channel capacity by *time division multiplexing* (TDM). These tables are used to set up *pipelined virtual circuits* and divide bandwidth between flows [19]. The circuits are pipelined in the sense that if a flow reserves slot s on a channel, then slot $s + 1$ must be reserved to the same flow on the succeeding channel in the path. The same slot table size $|S|$ is used throughout the entire network.

The realisation of the applications on the interconnection network is determined by the mapping function $map : P \rightarrow N_{NI}$ that maps core ports to NI ports, and the binding of flows to paths and time slots, captured in *configurations*. Existing work use the same temporal granularity of reconfiguration, determined by the compound modes, for all applications. Because of this, going from a use-case u_i to

u_{i+1} in Figure 2(b), is either done without any change (for the NoC) or requires global reconfiguration. An important contribution of this work is the specification of *configuration units* on a per application basis.

Definition 3. The *configuration units* Υ_a of an application a determine the temporal granularity of reconfiguration. Υ_a is a partition of the set of use-cases containing a, $U_a = \{u_0^a, \ldots, u_n^a\}$, into *jointly exhaustive* and *mutually exclusive* subsets. Applications that are *persistent*, with no disruption in QoS allowed, have $\Upsilon_a = \{U_a\}$ whereas *non-persistent* applications have $\Upsilon_a = \{\{u_0^a\}, \ldots, \{u_n^a\}\}$.

The partitioning Υ_a allows any granularity, but to simplify specification for the user, we only classify applications as either non-persistent or persistent. The latter is exemplified in Figure 2(c) where all applications have a single configuration unit.

A *configuration* $c : F \times \Upsilon_a \rightarrow \operatorname{seq} E \times \mathcal{P}(S)$ associates each flow with a path π and a set of time slots for a given configuration unit. The time slots are given relative to $head\ \pi$ but are reserved on the entire path. Together with the mapping function, the aforementioned function is refined as the algorithm progresses, i.e. initially no core ports are mapped to NIs and no paths or time slots are allocated. For notational clarity we refrain from subscripting and refer the reader to [9] for details on the successive refinement and proof of algorithm termination.

The contribution of the proposed model is shown in Figure 5, illustrating the transition from use-case u_1 to u_2 with and without partial reconfiguration. With configuration on the granularity of the entire NoC [16], the transition goes via the empty configuration $\varnothing$, and all flows must be stopped which requires a time that is unknown or difficult to bound as the flows must reach a *quiescent* state [12]. With partial reconfiguration, the common applications are unchanged.

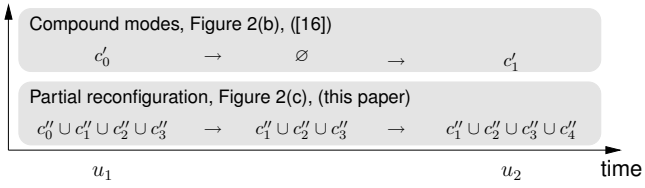


Figure 5. Instantiated configurations.

The general definition of configuration units does not exclude the possibility of having the same partitioning across all applications. The methodologies of [15,16] are thus subsumed in this more general framework.

5 Unified Mapping and Configuration

The objective of the NoC design flow [8] is to design the minimal NoC that satisfies the design constraints of all the use-cases. To accomplish the latter we enclose the proposed mapping and configuration algorithm in a topology selection loop where a cost function decided by the SoC designer is used to assess the cost, e.g. silicon area or power consumption, of the solutions.

We use a heuristic algorithm as already the problem of graph mapping with *static* routing and *without* TDM slot assignment is NP-hard [10]. In our approach, the mapping process is combined with the NoC configuration, resulting in quick pruning of the solution search space [9]. The core of the proposed algorithm for mapping of multiple applications onto a NoC is outlined in Algorithm 5.1 and introduced here, whereafter further explanations follow in Sections 5.1 and 5.2.

The overall idea is to: 1) Choose the most critical flow that is not yet assigned a path and time slots and potentially has source and destination ports not yet mapped to an NI, 2) derive the resources that are available in all use-cases spanned by the flow, and 3) allocate resources such that QoS requirements are met and then distribute the reservation across the data structures of the affected use-cases.

The body of the algorithm is iteration over the monotonically decreasing set of unallocated flow and configuration unit pairs $F' = \bigcup_{a \in A} F_a \times \Upsilon_a$. Note, that mapping and configuration is done across all use-cases in parallel [16]. In every iteration the most critical flow f is allocated in its corresponding configuration unit υ. We never backtrack to re-evaluate an already allocated flow, resulting in low time complexity at the expense of optimality.

First, a path (that also determines the core port to NI port mapping [9]) and a set of time slots are selected for f in Step 2a. Then in Step 2b, the mapping map and configuration $c(f, \upsilon)$ are refined to reflect the new state. The procedure is repeated until all flows are allocated. The resulting algorithm has polynomial time complexity with a run time dominated by the path selection algorithm. Note that once a solution is found the design space can be explored further by swapping vertices [14].

Algorithm 5.1 Allocation of all flows F

1. Let the set of unallocated flows $F' := \bigcup_{a \in A} F_a \times \Upsilon_a$ sorted in decreasing $criticality(f, \upsilon)$
2. While $F' \neq \varnothing$:
   (a) Select a path in $\Pi(s(f), d(f))$ and time-slots from the residual resources, $r(f, \upsilon)$, such that $t(f)$ and $l(f)$ are fulfilled
   (b) Refine $c(f, \upsilon)$ and map.
   (c) $F' := F' \setminus \{(f, \upsilon)\}$

5.1 Flow traversal order

The order in which flows are allocated in Algorithm 5.1, i.e. their *criticality*, takes two different measures into account. In order of priority: 1) the number of use-cases that the configuration unit of the flow spans and 2) an aggregate of the throughput and latency requirements [23].

The first heuristic asserts that flows spanning many use-cases are allocated early. Hence, the flows of the *resource manager* in Figure 2(c) are allocated first, followed by the *roam* flows. This ordering aims to reduce resource fragmentation between configurations as a configuration unit spanning several use-cases may only use time-slots that are not reserved in *any* of the associated configurations.

The secondary sort key ensures that flows with low throughput requirements but with tight latency constraints are given priority over those with high throughput requirement and relaxed latency constraints.

5.2 Path selection

Evaluation and selection of QoS constrained paths in Step 2a of Algorithm 5.1 is done by A*Prune [13]. We use path length as the optimisation criterion as power consumption scales with the hop count [11,23]. Path pruning is based on time-slot availability [9] to assert that performance guarantees are fulfilled [6].

When allocating a flow f in a configuration unit υ, the function $r : F \times \Upsilon_a$ returns the intersection between residual resources across all $u \in \upsilon$. The outcome is translated into an interconnection graph with a slot table per channel. The spatial and temporal routing algorithm uses this graph, and hence needs no knowledge about the multiple use-cases. A resource in the graph is unoccupied only if it is not reserved in *any* of the use-cases in υ. That is, $r(f, \upsilon)$ returns resources not reserved in any configuration $c(f', \upsilon')$ where the two configuration units υ and υ' overlap in time, $\upsilon \cup \upsilon' \neq \varnothing$. For example, when allocating a flow of the application *mp3 play* in Figure 2(c) only slots not yet reserved in c''_0, c''_1, c''_2, c''_3 and c''_4 are available.

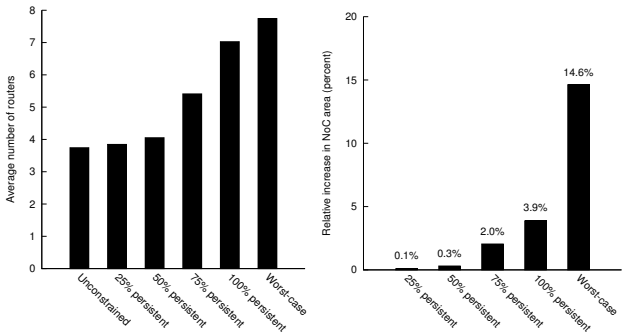
6 Experimental results

To evaluate the performance of our methodology, we apply it to a mobile phone SoC design and a range of synthetic benchmarks. The latter are structured in two classes, following the communication patterns of real SoCs. First, spread communication, representing SoCs with on-chip memories and point-to-point communication. Second, bottleneck communication, characterising designs with shared off-chip memory, involving a few cores in most communication.

NI buffer sizes are derived using analytical models [6] and silicon area requirements are based on the model presented in [7], assuming a 0.13 μm CMOS process. Total NoC area is used as the optimisation criterion.

6.1 Synthetic benchmarks

To evaluate the performance of our proposed methodology we compare three methods. First, the algorithm presented in [16], referred to as *unconstrained*, which uses a global configuration per use-case. Second, the methodology presented in this paper. Third, the synthetic worst-case approach introduced in [15], denoted *worst-case*. To show



(a) Average router count (b) Relative NoC area

Figure 6. Uniformly spread communication.

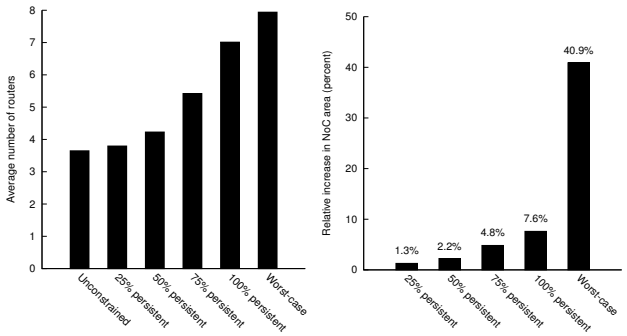
the impact of the number of persistent applications we generate four different NoCs with partial reconfiguration. One where 25% of the applications are persistent, one with 50%, one with 75% and lastly one design where all applications are persistent. Note that only that latter and the worst-case approach deliver undisrupted QoS.

The NoC is operating at 500 MHz with a link width of 32 bits. All benchmarks have 40 cores with a master and slave port. For every benchmark 16 applications are generated with 10-30 flows each. Bandwidth and latency requirements are varied across 4 bins respectively. This reflects for example a video SoC where video flows have high bandwidth requirements, audio have low bandwidth needs, and the control flows have low bandwidth needs but are latency critical. The applications are combined into use-cases according to a randomised graph, as in Figure 4. Half of the possible edges are present, with at most 8 applications allowed to run at once. A total of 100 benchmarks are generated per communication pattern. For all the benchmarks, the different methods produce the results in a few minutes when run on a Linux workstation.

Figure 6 shows the result of applying the three methods to benchmarks with uniformly spread communication. The router count increases with the addition of constraints, as more resources are needed to reduce contention. Looking at the total NoC area, dominated by the buffering requirements [7], we also see an increase with the number of constraints. Providing all applications undisrupted QoS increases the total NoC area by 4%. This is still 10% cheaper than the alternative approach of mapping the synthetic worst-case onto a NoC.

In Figure 7 the corresponding results for bottleneck communication are shown. The increase in area for the NoC when all applications have undisrupted QoS is below 8%. This is to be compared with a 40% increase for the alternative approach using a synthetic worst-case.

The different methodologies have different requirements on the amount of memory needed to store the configurations. For the worst-case and partial reconfiguration approach only one configuration needs to be stored per flow, resulting in a mere 1.5 kB (40 bits per flow and about



(a) Average router count (b) Relative NoC area

Figure 7. Bottleneck communication.

300 flows). With one configuration per use-case [16], this amount scales with the number of use-cases, resulting in several MBs of configuration data.

6.2 Mobile phone SoC

A phone SoC with telecom, storage, audio/video decoding, camera image encoding, image preview and 3D gaming constitutes our design example. The system has 13 cores (27 ports distributed across an ARM, a TriMedia, two DSPs, a rendering engine etc.), one off-chip DDR memory, one on-chip SRAM plus a number of peripherals. Communication is done via memory, running at 117 MHz with a word width of 64 bits. As the native word width of our NoC is 32 bits we choose to let the NoC run at double the frequency, 235 MHz, thus offering the same gross bandwidth.

We compare the NoC architecture generated by the unconstrained mapping, partial reconfiguration with 100% of the applications persistent, and the worst-case approach. Table 1 shows the resulting NoC designs for the three methods, optimising on total area (mm^2). While the worst-case methodology increases the NoC area with more than 23%, the methodology introduced in this paper delivers undisrupted application QoS with an area increase below 2%. Similarly for the power consumption, calculated according to the model in [4], the average increase across all use-cases is 129% for the worst-case methodology whereas it is only 3% with partial reconfiguration. For all use-cases the power consumption of our methodology is less than half of that of the worst-case methodology. This stems from having a network with half the amount of routers and shorter paths.

Table 1. Comparison of methodologies.

Methodology	Mesh	Slots	NI area	Router area	Total area	Area diff	Power diff
unconstrained	1×3	29	1.8	0.2	2.0	ref	ref
100% persistent	1×3	29	1.9	0.2	2.1	+2%	+3%
worst-case	2×3	15	2.0	0.5	2.5	+23%	+129%

From the experiments we conclude that the area cost incurred by resource fragmentation is only a few percent, even when all applications are persistent. Compared to the alternative worst-case approach, this work results in NoC designs that are considerably smaller.

7 Conclusion and future work

In this paper we present a model that enables partial reconfiguration of Networks on Chip (NoC) and an algorithm that uses the model to map multiple applications onto a NoC, delivering undisrupted Quality-of-Service during reconfiguration. By distinguishing between different *spatial* as well as *temporal* granularities of reconfiguration, the different applications are associated with their own virtual resources and can be removed or added independently.

The performance of the methodology is verified by comparison with existing solutions for several SoC designs. We apply the algorithm to a mobile phone SoC with applications like telecom, multimedia and gaming, reducing NoC area by more than 17% and power consumption by more than 50% compared to a state-of-the-art design approach.

Future work includes evaluating the proposed methodology in combination with a run-time configuration manager.

References

[1] L. Benini and G. de Micheli. Networks on chips: A new SoC paradigm. *IEEE Comp.*, 35(1), 2002.
[2] D. Bertozzi *et al.* NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Trans. on Par. and Distr. Syst.*, 16(2), 2005.
[3] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. DAC*, 2001.
[4] J. Dielissen *et al.* Power measurements and analysis of a network-on-chip. Technical Report NL-TN-2005-0282, Philips Research Laboratories, 2005.
[5] S. Dutta *et al.* Viper: A multiprocessor SOC for advanced set-top box and digital TV systems. *IEEE Des. and Test of Comp.*, pages 21–31, 2001.
[6] O. P. Gangwal *et al. Dynamic and Robust Streaming In And Between Connected Consumer-Electronics Devices*, chapter 1. Kluwer, 2005.
[7] S. González Pestana *et al.* Cost-performance trade-offs in networks on chip: A simulation-based approach. In *Proc. DATE*, 2004.
[8] K. Goossens *et al.* A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification. In *Proc. DATE*, 2005.
[9] A. Hansson *et al.* A unified approach to constrained mapping and routing on network-on-chip architectures. In *Proc. CODES+ISSS*, 2005.
[10] J. Hu and R. Mărculescu. Energy-aware mapping for tile-based NoC architectures under performance constraints. In *Proc. ASP-DAC*, 2003.
[11] J. Hu and R. Mărculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures. In *Proc. DATE*, 2003.
[12] J. Kramer and J. Magee. The evolving philosophers problem: Dynamic change management. *IEEE Trans. on Soft. Eng.*, 16(11), 1990.
[13] G. Liu and K. G. Ramakrishnan. A*Prune: An algorithm for finding K shortest paths subject to multiple constraints. In *Proc. IEEE INFOCOM'01*, 2001.
[14] S. Murali *et al.* Mapping and physical planning of networks on chip architectures with quality of service guarantees. In *Proc. ASP-DAC*, 2005.
[15] S. Murali *et al.* Mapping and configuration methods for multi-use-case networks on chips. In *Proc. ASP-DAC*, 2006.
[16] S. Murali *et al.* A methodology for mapping multiple use-cases on to networks on chip. In *Proc. DATE*, 2006.
[17] V. Nollet *et al.* Centralized run-time resource management in a network-on-chip containing reconfigurable hardware tiles. In *Proc. DATE*, 2005.
[18] A. Pinto *et al.* Efficient synthesis of networks on chip. In *Proc. ICCD*, 2003.
[19] E. Rijpkema *et al.* Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. *IEE Proc. Comp. and Dig. Techn.*, 150(5), 2003.
[20] M. Rutten *et al.* Dynamic reconfiguration of streaming graphs on a heterogeneous multiprocessor architecture. *IS&T/SPIE Electron. Imag.*, 5683, 2005.
[21] M. Sgroi *et al.* Addressing the system-on-a-chip interconnect woes through communication-based design. In *Proc. DAC*, 2001.
[22] L. T. Smit *et al.* Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture. In *Proc. FPT*, 2004.
[23] K. Srinivasan *et al.* An automated technique for topology and route generation of application specific on-chip interconnection networks. In *Proc. ICCAD*, 2005.