Demonstration of Run-time Spatial Mapping of Streaming Applications to a Heterogeneous Multi-Processor System-on-Chip (MPSOC)

Article · January 2008

DOI: 10.1145/1403375.1403427 · Source: OAI

4 authors, including:

Philip K.F. Hölzenspies
Facebook
30 PUBLICATIONS **261** CITATIONS
SEE PROFILE

Jan Kuper
University of Twente
87 PUBLICATIONS **392** CITATIONS
SEE PROFILE

Johann Hurink
University of Twente
203 PUBLICATIONS **2,521** CITATIONS
SEE PROFILE

Run-time Spatial Mapping of Streaming Applications to a Heterogeneous Multi-Processor System-on-Chip (MPSoC)

Philip K.F. Hölzenspies, Johann L. Hurink, Jan Kuper, Gerard J.M. Smit
University of Twente
Department of Electrical Engineering, Mathematics and Computer Science
P.O. Box 217, 7500 AE Enschede, The Netherlands
p.k.f.holzenspies@utwente.nl

Abstract

In this paper, we present an algorithm for run-time allocation of hardware resources to software applications. We define the sub-problem of run-time spatial mapping *and demonstrate our concept for streaming applications on heterogeneous MPSoCs. The underlying algorithm and the methods used therein are implemented and their use is demonstrated with an illustrative example.*

1. Introduction

In this paper we present an algorithm for the run-time mapping of streaming applications onto a MPSoC. Such a mapping has to allocate sufficient resources to the application before the application can start. For embedded and/or energy-constrained devices, it is important that this allocation is done such that as few as possible energy is consumed.

In conventional *uniprocessor* systems, schedules for applications and their resource requirements can be optimized exhaustively at design-time. In *homogeneous* multiprocessor systems, such a design-time optimization gets more complex since a spatial choice is *added to* the temporal choice of resources; applications can share a resource and wait on each other, or they can run simultaneously with independent resources. This implies that the optimization can only be done heuristically, since exhaustive search already requires far too much time. In *heterogeneous* systems (tiled architectures) the resource allocations becomes even more complex. Each application has a preferred type of resource (e.g. a DSP) on which it ideally should be allocated. But it can, if necessary, run with another type of resource (e.g. an ARM) as well (albeit less efficiently). Moreover, often a full design-time exploration is not even possible, since not all software a user may run on a system is known at design-time.

Another drawback of design-time exploration lies in the fact that many modern systems allow a wide variety of usage scenarios (different combinations of applications, different Quality of Service (QoS) constraints), all of which have different optimal resource allocations. On the one hand, the cost of finding optimal solutions for all these scenarios, storing these solutions for run-time reference poses considerable problems. On the other hand, design-time explorations are generally based on worst-case assumptions with respect to resource requirements of applications. These may be far away from actual requirements when applications are started (e.g. the computational resources required for baseband processing depend heavily on channel conditions).

By moving resource allocation out of design-time to the moment when applications are started, we introduce a higher degree of flexibility and can thus incorporate run-time information to further reduce the (energy) cost of running the application. Deriving proper methods for such a run-time allocation is the challenge tackled in this paper. In the remainder of this section we present in more detail the context in which we treat the problem and we introduce the concepts and terminology.

1.1. Tiled architectures

As heterogeneous system we consider a *tiled architecture*. In this paper we understand this as a composable system made up out of multiple processing elements. The combination of a processing element and its interface to the architecture's interconnect is referred to as a *tile*. Furthermore, we assumes that the tiles on the chip are interconnected by a predictable (with respect to throughput and latency[5]) Network-on-Chip (NoC).

1.2. Streaming DSP applications

The concept of run-time mapping fits mainly to long running applications. Therefore, we restrict ourself to streaming DSP applications. Such applications consist of (multiple) computational kernels, operating on a stream of data. Many of these kernels can be implemented on very specialized hardware as well as on more general purpose architectures. We assume to have multiple implementations for

many of the kernels. Generally speaking, streaming DSP applications have a predictable behaviour, both temporally and spatially and can be modelled by task graphs[3, 9]. Typical examples are found in signal processing for wireless baseband processing (for wireless LAN, digital radio, UMTS), multi-media processing, medical image processing and sensor processing.

On a functional level, a streaming application can be described as a Kahn Process Network (KPN), because only the functional decomposition of an application and the data dependencies between the kernels is specified. Concrete implementations of these kernels need to be specified with much more detailed information, i.e. Worst-Case Execution Time (WCET) and granularity of consumption and production of data (e.g. an implementation of a function on video frames may read the entire frame before executing, but it may also work only on the first few lines).

A fine grained specification by means of Cyclo-Static Data Flow [2] (CSDF) graphs is used to allow for detailed analysis[11]. In CSDF, actors and edges are labelled respectively with WCET and token production and consumption rates, for all different phases of execution of the actor.

1.3. Run-time spatial mapping

Generally, spatial mapping is the allocation of spatial resources for applications. In the context of tiled architectures, spatial resources are tiles and—in the case of a NoC—routers. Thus, spatial mapping is the assignment of tasks from the KPN, describing the streaming application, to tiles and channels to paths through the NoC. A *feasible* spatial mapping satisfies the mapped application's QoS requirements. A spatial mapping's *quality* depends on the extend to which it minimizes cost (in our case: energy consumption) under the resource constraints. The objective of run-time spatial mapping is to find a feasible spatial mapping with the best quality (in our case: the lowest energy cost).

To be able to utilize heterogeneous multi-tile systems efficiently, tasks that are used often in streaming applications should be implemented for different tile types. For example, for a frequently used DSP kernel such as an FFT, there are implementations for an embedded ARM tile and for a reconfigurable core. Thereby, flexibility is introduced that allows a task to be executed, even if there is no tile available of the most preferred type, e.g. when all tiles of the preferred type are occupied.

This flexibility furthermore allows for software to be introduced by the user, unknown at the time the system was designed. Since the availability of resources depends on the set of applications running simultaneously and variations in QoS requirements due to changes in the environment, a design-time mapping requires worst case assumptions. However, at run-time when starting an application, the actual set of applications already running is known, allowing for a spatial mapping based on actual, rather than worst case information. As a result, the mapping generated at run-time may actually be cheaper than the cheapest design-time alternative.

Goals and requirements In our context, the objective of the spatial mapping is to minimize the energy consumption of the entire application: processing (including memory requirements thereof) as well as interprocess communication. In principle, the spatial mapping is performed always when a new streaming application is started. The run-time setting asks for fast and simple methods. To be able to perform the mapping of an application to tiles, a spatial mapping algorithm needs models of the application and of the MPSoC. Furthermore, the QoS constraints of the application (e.g. throughput requirements and latency bounds) need to be known, as well as the resource requirements of the available implementations (e.g. time, memory, etc.)

1.4. Overview

The remainder of this paper is structured as follows. Section 2 describes the contributions made by this paper and the related work. An algorithm implementing our initial spatial mapper is described in Section 3. To provide some intuition, a full case example is given in Section 4, which also includes a brief summary of the experimental implementation we made of the spatial mapper. Conclusions are drawn in Section 5.

2. Contribution & Related Work

Moreira et al. have demonstrated methods for user initiated on-line resource management [8]. By means of a vector bin packing algorithm, optionally preceded by clustering of neighbours, they can map tasks from task graphs to tiles and interconnect. However, their method relies on the homogeneity of the processors and aims to minimize the number of off-tile connections, because network connectivity is scarce in their system (three tiles per five port router) and clustering works well only in homogeneous systems.

Kumar et al. have shown an on-line resource management for heterogeneous multi-tile systems [6]. They demonstrate quite clearly that information unavailable until run-time can actually improve utilization of resources and throughput of applications. However, they only consider temporal settings at run-time, i.e. an assignment of tasks to tiles is made at design-time.

A lot of work has also been done for mapping applications to FPGAs at run-time, e.g. Banerjee et al. [1]. However, since FPGAs are reconfigured on a per-column basis, not only is the problem inherently homogeneous, but since reconfiguration often affects communication, the focus lies on exploiting data parallelism with tasks and reconfiguring between tasks (i.e. multiplexing tasks in time rather than space).

We propose heuristics to perform the mapping at a task-level granularity taking into account the tight time constraints that come with the run-time setting. By separat-

ing the spatial and temporal mappings, we have achieved promising results[10].

3. Algorithm

Spatial mapping is a complex problem. Even when only considering the assignment of processes to a heterogeneous multi-tile platform, we find a Generalized Assignment Problem [7] (GAP), which is NP-complete. Considering the prohibitive complexity of exhaustive search, we propose an application domain aware heuristic: *hierarchical search with iterative refinement*. We divide the search process in steps, starting with a very coarse perspective in the first step and gradually adding more detail. At each step decisions are made that shrink the search space in the next. Decisions made in previous steps are considered fixed in later steps.

As is to be expected of heuristics, this abstraction carries with it the danger that decisions made in early steps, using very high level abstract information, lead to search spaces in later steps that contain no feasible solutions. Since this infeasibility only comes to light in later steps, we propose a strategy for iterative refinement (we rerun the steps as needed). In the following we describe each of these steps in more detail.

For ease of explanation, we define three characteristics for spatial mappings. A spatial mapping is considered *adequate* if for all processes there is an implementation available for the type of tile to which it is assigned. A mapping is considered *adherent* when it is adequate and there are no tiles that are assigned more processes than they are able to serve. Finally, a mapping is considered *feasible* if it is adherent and all the application's constraints are met.

1. **Assign implementations to processes.** The goal of the first step is to choose an implementation (and thereby tile type) for every process. To prevent running into inadherence directly after this step, we only consider those implementations for which an adhering mapping exists, i.e. such that every implementation fits on at least one tile in the system.

   We go about this choice iteratively. The choice of the next process to pick an implementation for is based on its *desirability*. The desirability of a process is the difference between the cheapest assignment and the second cheapest assignment of the process to a tile. In other words, if the alternative is more expensive, the desirability to map the process 'now' increases.

   If a process has been chosen to be assigned next, not only do we choose this process' implementation, we also map it to the first tile we come across with sufficient resources (i.e. a first-fit packing). This guarantees that after this step (if this step manages to map all processes), at least one concrete tile assignment exists that does not break the adherence of the mapping, although the mapping might still be inadherent when the communication can not be mapped successfully.

2. **Assign processes to tiles.** The (greedy) assignment of processes to specific tiles obtained in the previous step is now improved upon by taking more detail into account. In a local search type fashion, we iteratively make a choice, based on desirability. In particular, for every implementation we try to remove it from the tile it is mapped onto and to map it onto the best available tile of the same type. Alternatively, we try to swap the process with another process mapped to the same tile type. The difference in cost between the original mapping and the best tile found is, again, the measure of desirability for choosing this reassignment. Only the best reassignment is actually performed every iteration.

   This step uses heuristics to look ahead towards communication, but does not have exact knowledge of the status of the complete NoC. Improvements arise from having to communicate less (probably, since exact routing is not known here) and from being able to turn off parts of the system that are not being used. The Manhattan distance is used to estimate how much a channel's communication costs. The total communication cost of assigning an implementation to a tile is the sum of the Manhattan distances of all the implementation's incoming and outgoing channels. Since a process can only be reassigned to a tile with the same type as the tile it is already assigned to, this step maintains adequacy by construction.

   Step one simply iterates until all processes are assigned to a tile (one process per iteration). Deciding when to stop in step two can be based upon a minimum gain from the current iteration (once an iteration improves the total solution by a lesser amount than a chosen threshold, we decide to stop) and/or by a maximum number of iterations. Besides cost factors based solely on the mapping of a process to a tile, an assignment should be awarded a bonus for proximity to the process' neighbours in the application graph. This stimulates locality, causing the communication routes, assigned in the next step, to likely be short. Moreover, we again prevent immediate inadherence in the next step, by only considering tiles for a process that have sufficient communication resources to facilitate the process' communication requirements, at least, locally.

3. **Assign channels to paths.** For the concrete realization of step three, the channels are sorted by non-increasing throughput. Next, iteratively for each channel, a corresponding path is determined, taking into account the loads resulting from the previously mapped channels. The sorting is done to increase the probability that a heavy demanding channel gets assigned a better path. In each iteration for a given channel, a shortest path between the source and destination tile of the channel has to be determined, where only those paths through the interconnect are taken into account which still have enough capacity for the throughput requirement of the current channel.

This method assures that all channels are mapped onto paths through the interconnect that can guarantee the required throughput. Adding this mapping of the channels to the mappings from the previous steps results in an *adherent* spatial mapping.

4. **Check application constraints.** The last step checks the QoS constraints. When any such constraint is violated, the spatial mapping is *infeasible* and feedback should be given to earlier steps to try and improve upon those characteristics of the mapping that violate the constraint(s). Should no constraint be violated, the spatial mapping is *feasible*. We use a data flow analysis for this check, that is beyond the scope of this paper. Instead, we reference [11]. When we decide we have time to look for improvements of this solution, possible points of improvement should also be identified here and fed back into the first step (keeping the current mapping in mind, should the feedback only result in infeasible results and feasible mappings that are worse than the current best solution).

In general, the production of feedback immediately triggers a new iteration, to prevent that multiple changes influence the mapping process. In other words, if any step fails to find a satisfactory result, it immediately generates feedback so that 'higher' steps may generate a more suitable result.

It is important to realize that this proposed iterative hierarchical approach differs significantly from simple local search methods and global-local search methods that are often used in heuristics. The feedback from a lower level may result in a completely different mapping on a higher level in a next iteration.

4. Case: HiperLAN/2

In order to illustrate the above with an example, an implementation and mapping of a HiperLAN/2 receiver is described in this section.

4.1. Application Level Specification

The receiver's decomposition into communicating processes is shown in the KPN in Figure 1. The control part of the receiver application is included for completeness, but it is not part of the data stream. Orthogonal Frequency Division Multiplexing (OFDM) applications are based on (MAC) frames of OFDM-symbols, which, in turn, consist of samples (complex numbers). In the HiperLAN/2 case, frames consist of 500 symbols and every symbol consists of 80 32-bit complex numbers. The control part only comes into play briefly at the beginning of each frame, while all other processes in the KPN operate on every OFDM symbol.

The last three processes have been grouped to form one process. Not only do they fit well together in a single implementation, but treating them separately needlessly lengthens this example. The numbers shown on the edges of this KPN indicate the number of 32-bit complex numbers per

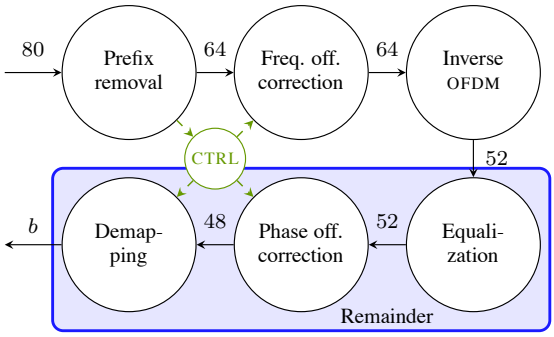

Figure 1. Decomposition of a HiperLAN/2 receiver

symbol coming in at each process. The size of the output of the HiperLAN/2 receiver (b), depends on what 'mode' the receiver is in. The standard defines seven modes, that only differ with regards to the demapping (hence the input from the control process, which selects the demapping type). Depending on the chosen demapping type, the output can be between 2 bits (BPSK) and 64 bits (QAM64) per sample. Thus the minimum output is 12 bytes and the maximum is 384 bytes (per OFDM symbol). One OFDM symbol is fed into the application once every $4\mu s$.

The graph describing functional dependencies of the processes and the QoS constraints together form the Application Level Specification (ALS).

4.2. Implementations

Given a set of implementations of the processes in Figure 1, the spatial mapping algorithm can now choose implementations, map them to concrete tiles, route the communication channels through the interconnect and construct a CSDF graph of the entire receiver. The description of any implementation should include a CSDF graph, describing its behaviour correctly and in as much detail as is relevant. As stated above, many processes can be described as a single CSDF actor. Table 1 lists implementations of the processes in Figure 1, where the numbers come from design-time analyses and profiling. The phases described in the table are the phases of the CSDF actors corresponding to the implementations. In this table the notation for the inputs $\langle 0, a, b^{10} \rangle$ means: in phase 1, 0 tokens are read, in phase 2, a tokens are read, and in phase 3 through 12, b tokens are read at every phase (the superscript is a shorthand for the number of phases with equal parameters). For example: the inverse OFDM on the ARM has 3 phases; in phase 1, 64 tokens are read, 0 tokens are written and the WCET is 66 clock cycles; in phase 2, 0 tokens are read, 0 tokens are written and the WCET is 4250 cc; in phase 3, 0 tokens are read, 64 tokens are written and the WCET is 54 cc.

Control-flow has been omitted from this table, but *will* be taken into account in the verification process. Note that

Table 1. Available implementations

Process	PE type	Phases[a] Input [token]	Phases[a] Output [token]	Phases[a] WCET [clockcycles]	Avg. energy $[nJ/\text{symbol}]$
Prefix removal	ARM	$\langle 8^2, \langle 8,0\rangle^8\rangle$	$\langle 0^2, \langle 0,8\rangle^8\rangle$	$\langle 18^{18}\rangle$	60
	MONTIUM	$\langle 1^{80}, 0\rangle$	$\langle 0^{17}, 1^{64}\rangle$	$\langle 1^{81}\rangle$	32
Freq. off. correction	ARM	$\langle 8,0,0\rangle$	$\langle 0,0,8\rangle$	$\langle 18,32,18\rangle$	62
	MONTIUM	$\langle 1^{64}, 0^2\rangle$	$\langle 0^2, 1^{64}\rangle$	$\langle 1^{66}\rangle$	33
Inverse OFDM	ARM	$\langle 64,0,0\rangle$	$\langle 0,0,64\rangle$	$\langle 66,4250,54\rangle$	275
	MONTIUM	$\langle 1^{64}, 0^{53}\rangle$	$\langle 0^{65}, 1^{52}\rangle$	$\langle 1^{64}, 170, 1^{52}\rangle$	143
Remainder	ARM	$\langle 52,0,b\rangle$	$\langle 0,0,b\rangle$	$\langle 54,2250,b+2\rangle$	140
	MONTIUM	$\langle 1^{52}, 0, 0$	$\langle 0,0,1^b\rangle$	$\langle 1^{52}, 73-b, 1^b\rangle$	76

[a] We use the notation $\langle x^n, y^m\rangle$ to denote $n+m$ phases, where the value for the first n phases is x and for the last m phases is y.

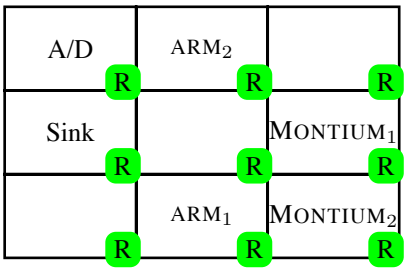

Figure 2. MPSOC **layout**

the input and output token counts are in terms of samples (i.e. one token corresponds to one complex number). The execution times given in the table are in terms of clock cycles.

4.3. Hardware

A few notes are required on the hardware of the test environment. The HIPERLAN/2 receiver is mapped to a MPSOC, consisting of ARMs with cache and a coarse grain reconfigurable type of tile, namely MONTIUMs [4].

The interconnect consists of a NOC with routers that provide guarantees with regards to provided throughput and maximum latency[5]. All tiles have their own Network Interface (NI) to connect to the NOC, but the NOC side of that interface is the same for every tile. The routers in the NOC have buffered inputs and round-robin arbitration on the output, which imposes a maximum latency of 4 clock cycles. For brevity, a homogeneous NOC is considered here, implying that every step in a communication path has the same behaviour and, thus, the same description in the CSDF graph.

The hypothetical MPSOC used for this example has two MONTIUMs and two ARMs. Figure 2 shows a possible MPSOC layout with these specifications. The tiles without labels in this figure are tiles of types not relevant to this example. The tile labelled 'A/D' is the source of all the incoming data. The tile labelled 'Sink' is the tile that has to receive the stream flowing out of the HIPERLAN/2 receiver.

4.4. Mapping

We go about the mapping as described in Section 3. In this example, the 'Inverse OFDM' process is the most desirable. Thus, it is assigned to its preferred tile type, being a MONTIUM. Likewise, the 'Remainder' process is assigned a MONTIUM. At this point, both MONTIUMs are occupied and thus, the available implementations for the MONTIUM architecture can no longer be assigned to a tile. This means that all these implementations are ignored from here on. As such, both remaining processes only have ARM implementations and are thus chosen per default.

In the second step, the chosen implementations need to be assigned to specific tiles. Table 2 shows the iterations of the algorithm in step two, that lead to the final implementation-to-tile assignment. Swaps can, of course, only occur between tiles of the same type. The sum of all Manhattan distances of the application (the cost column) can increase or remain the same for any iteration. When this happens, that choice is rejected and another is evaluated.

As a last step in the mapping process, step three performs incremental routing. This means the channels from the ALS are routed incrementally with a point-to-point shortest path algorithm. Only those lanes in the NOC that can guarantee sufficient throughput are considered. Figure 3 shows the resulting CSDF graph. This graph can be checked (with [11]) to see whether the mapping suffices to meet the QOS constraints laid down in the ALS (step 4 of the spatial mapper). The buffer sizes B_i are calculated by the algorithm used in step 4. When they are smaller than the buffers reserved by the implementations, no further action is required. When they are larger, an attempt should be made to allocate the

Table 2. Processor assignment iterations in step 2

	ARM		MONTIUM			
Iter.	1	2	1	2	Cost	Remark
-	Pfx.rem.	Frq.off.	Inv.OFDM	Rem.	11	Initial (greedy) assignment
1	**Frq.off.**	**Pfx.rem.**	Inv.OFDM	Rem.	11	No improvement, revert
2	Pfx.rem.	Frq.off.	**Rem.**	**Inv.OFDM**	9	Improvement, keep
3	**Frq.off.**	**Pfx.rem.**	Rem.	Inv.OFDM	7	Improvement, keep
						No further choices

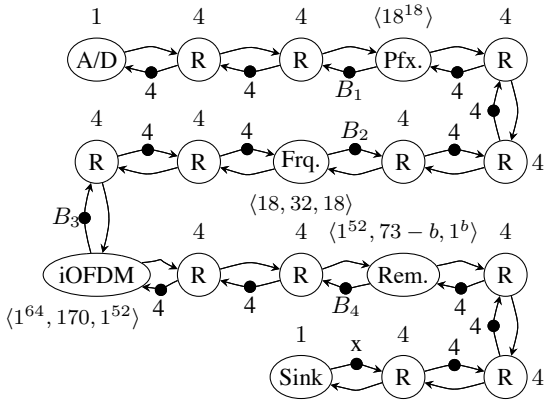

Figure 3. Final CSDF graph (production and consumption rates omitted to prevent clutter)

additional required buffer size on the tiles the consuming actor is mapped onto. If this additional buffer capacity can not be allocated, the mapping is inadherent and the spatial mapper should iterate. The buffer capacity of the Sink actor (x) is fixed by the specification of Sink.

4.5. Implementation

We have implemented the algorithm on an ARM. The total code size of the implementation (compiled with GCC 3.2.1 eCosCentric, no explicit optimization switches) is 137 kB. Running the HIPERLAN/2 example through it on an ARM926 running at 100 MHz took less then 4 ms. Peak data memory usage was 110 kB.

5. Conclusions and future work

We have presented an algorithm for run-time spatial mapping of streaming applications to MPSOCs. Criteria (adherence, adequacy and feasibility) were introduced for qualitative comparison of spatial mappings. Our algorithm to implement spatial mapping was described and shown to guarantee that solutions found are verifiable for feasibility.

To our knowledge, no benchmarks exist to compare spatial mappings quantitatively. These benchmarks must be developed. They should include far more complex real-life examples than the HIPERLAN/2 case used in this paper *and* synthetic cases based on the class of applications that can reasonably be expected for MPSOCs in the future.

In the current work, feedback can only be given based on the feasibility analysis of step four of the algorithm. When earlier steps fail to find a solution, feedback information should be produced with which a new attempt can be made.

References

[1] S. Banerjee, E. Bozorgzadeh, and N. Dutt. Parlgran: parallelism granularity selection for scheduling task chains on dynamically reconfigurable architectures. In *ASP-DAC '06: Proceedings of the 2006 conference on Asia South Pacific design automation*, pages 491–496, 2006.

[2] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cycle-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, 1996.

[3] W. J. Dally, U. J. Kapasi, B. Khailany, J. H. Ahn, and A. Das. Stream processors: Progammability and efficiency. *Queue*, 2(1):52–62, 2004.

[4] P. M. Heysters. *Coarse-Grained Reconfigurable Processors – Flexibility meets Efficiency*. PhD thesis, University of Twente, The Netherlands, 2004.

[5] N. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. PhD thesis, University of Twente, 2006.

[6] A. Kumar, B. Mesman, B. Theelen, H. Corporaal, and H. Yajun. Resource manager for non-preemptive heterogeneous multiprocessor system-on-chip. In *Proc. on Embedded Systems for Real Time Multimedia*, pages 33–38, Oct. 2006.

[7] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[8] O. Moreira, J. D. Mol, and M. Bekooij. Online resource management in a multiprocessor with a network-on-chip. In *Proc. of SAC '07*, pages 1557–1564, 2007.

[9] G. Smit, A. B. Kokkeler, P. T. Wolkotte, P. K. F. Hölzenspies, M. D. van de Burgwal, and P. M. Heysters. The chameleon architecture for streaming dsp applications. *EURASIP Journal on Embedded Systems*, 2007:78082, 2007.

[10] L. Smit, J. Hurink, and G. Smit. Run-time mapping of applications to a heterogeneous SoC. In *Proceedings of the 2005 International Symposium on System-on-Chip*, pages 78–81, Nov. 2005.

[11] M. Wiggers, M. Bekooij, and G. Smit. Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *Proc. of DAC '07*, pages 658–663, 2007.