Project no. 248828

ADVANCE

Strategic Research Partnership (STREP)
ASYNCHRONOUS AND **D**YNAMIC **V**IRTUALISATION THROUGH PERFORMANCE
ANALYSIS TO SUPPORT **C**ONCURRENCY **E**NGINEERING

Dynamic extension of placement heuristics

D10

Due date of deliverable: May 31st, 2011
Actual submission date: July 22st, 2011

Start date of project: February 1st, 2010

Type: Deliverable
WP number: WP6
Task number: WP6b

Responsible institution: TWENTE
Editor & and editor's address: Jan Kuper
University of Twente
Department of EEMCS
7500 AN Enschede, The Netherlands

Version 1.0 / Last edited by Robert de Groote / July 22, 2011

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
1.0	22/7/2011	Robert de Groote	TWENTE	Final version
0.8	15/7/2011	Robert de Groote	TWENTE	Section 2.3
0.4	29/6/2011	Robert de Groote	TWENTE	Major changes to chapters 2 and 3
0.2	13/5/2011	Jan Kuper	TWENTE	Revised document outline
0.1	15/4/2011	Robert de Groote	TWENTE	Initial version

Reviewers:
Jan Kuper, Robert de Groote

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
WP6b	Heuristics for dynamic placement	TWENTE*,UvA,USTAN,TECHNION

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Executive Summary

The goal of work package 6 is to design the interface between the previous levels in the tool chain and the chosen, often heterogeneous multi-core hardware platform. This includes the choice of the (near) optimal hardware from a set of hardware platforms, the dynamic (re-)placement of software modules on resources, the observation of run-time performance of the resulting system, and reporting the results of these observations back such that decisions made can be optimized.

The aim of task WP6b is to develop heuristics that may be used in the search for an optimal software to hardware mapping. These heuristics must take into account both computation and communication aspects. Different behaviours of software modules need to be taken into account, ranging from aperiodically started, short running functions to strongly periodic tasks such as functional repetitive processes. These heuristics exploit the structural monotoring information and resource usage models developed in WP3 and WP4. This report explores the main challenges involved in designing heuristics that exploit statistical profiles.

Contents

Chapter 1

Introduction

Mapping a software application to a heterogeneous hardware platform under performance constraints is a complex problem. The resource manager developed within WP6 splits this problem into separate phases. During each of these phases, a search is conducted. Heuristics may be used to guide these searches.

In a static resource management paradigm, information regarding the application's performance given a hardware mapping is restricted to worst-case performance information only. Dynamic resource management on the other hand takes the actual observed performance data into account. Within ADVANCE, this observed performance data is captured in resource usage models, which are given in the form of statistical profiles.

1.1 Goal of this task

The goal of this task (WP6b) is to develop heuristics to guide the search for an optimal placement on a hardware platform for a given application. Heuristics can be used in different phases of the search process. This task focuses on heuristics that exploit structural monitoring information and resource usage information developed in WP3 and WP4.

1.2 Relation with other work packages

Work carried out in work package 6 deals with run-time resource management and the development of the *resource management layer*. Interaction between the resource management layer and other layers is illustrated in figure 1.1. The resource management layer is responsible for analysing and evaluating the application's performance by observing the hardware pool. Statistical performance information that is compiled from the observed performance is fed back to higher levels through the hardware virtualisation layer.

Work package 6 is closely related to work package 4 in which static analysis for resource requirements is researched. Therefore, results obtained in this work

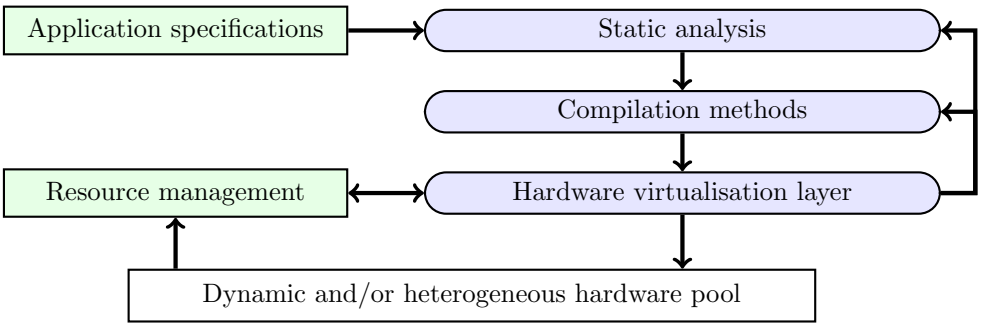


Figure 1.1: The Advance vision

package will find useful applications in WP4.

Smart resource management is a key issue in the X-ray image processing use case that is developed in work package 7. This use case involves an application that shows highly dynamic timing behaviour due to strong data dependent behaviour. At the time of writing, the other use cases are less developed but are expected to benefit from the experience gained in this work.

1.3 Outline

This report continues with a brief explanation of background concepts and heuristics used in the structural phase of the mapping. In chapter 2 several heuristics are discussed for a static mapping (in which case only worst-case performance information is known). These heuristics are taken as an input to the designing of heuristics that exploit observed information regarding timing behaviour in chapter 3. Finally, the report concludes with directions for future work within work package 6 in chapter 4.

Chapter 2

Heuristics in Resource Management

This chapter describes the heuristics used by the runtime resource manager "Kairos" that is used and further developed in ADVANCE. Kairos currently uses information that is known at design-time, such as worst-case execution times and memory requirements. Because Kairos assumes that applications that are mapped to a hardware platform may be modelled as synchronous dataflow (SDF) graphs, many of the extra challenges that arise in more dynamic applications are avoided. Instead of starting from scratch with designing resource management heuristics for applications that may be modelled by S-Nets, we believe that the targeted application model should be complexified gradually. This chapter therefore describes the factors taken into account in spatial resource management of data streaming applications, which serve as building blocks for heuristics guiding the mapping of applications with more dynamic behaviour and which are discussed in the next chapter. The resource manager "Kairos" is described in more detail in [15] and [23].

2.1 The mapping problem

An application defines a set of resources and performance constraints. Resource constraints may be minimum required amounts of available memory, requirements on the instruction set offered by a processing element, bandwidth requirements for communication between tasks, etc. Performance constraints may be specified as worst case bounds such as minimum throughput, maximum latency and / or maximum jitter, or may be 'softer' like quality of service specifications.

An application may be formally described as a task graph: a set of tasks and a set of communication channels that connect these tasks. Spatial resource management can be modelled as a *capacity planning problem*: the processing and communication capacity offered by the hardware platform is used by the application's tasks and should optimally be planned in such a way that resources are neither over- or under-utilised.

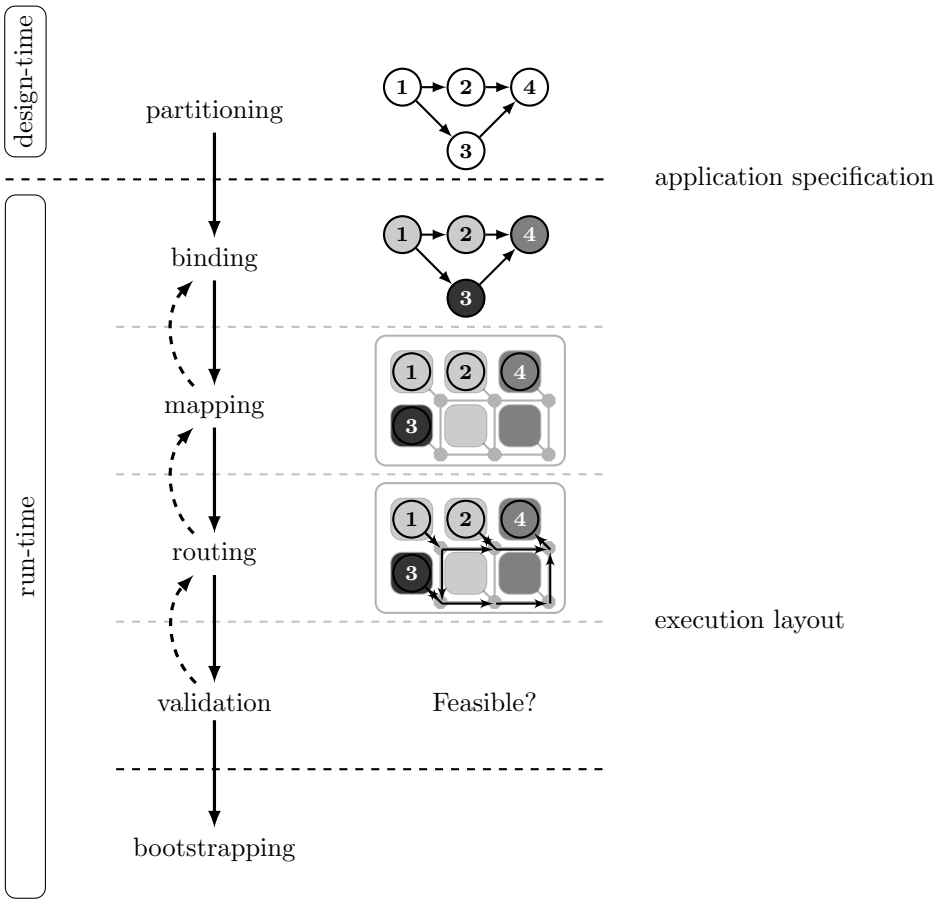


Figure 2.1: The different phases of the resource manager Kairos

2.2 Workflow

As finding an optimal solution to the mapping of an application to a hardware platform is a problem of high complexity, Kairos splits the problem into multiple subproblems, which are dealt with in separate phases. Figure 2.1 shows the resulting workflow where these phases are ordered from binding to validation. Each of the phases is described in the following sections.

2.2.1 Binding

Heterogeneous hardware platforms have hardware components with different capabilities. Each *processing* elements realises an Instruction Set Architecture (ISA). Processing elements with the same ISA are capable of executing the same tasks, not considering external restrictions such as limited memory capacities. The performance of a task may vary among processing elements with the same ISA.

During the binding phase an assignment of task implementations to processing

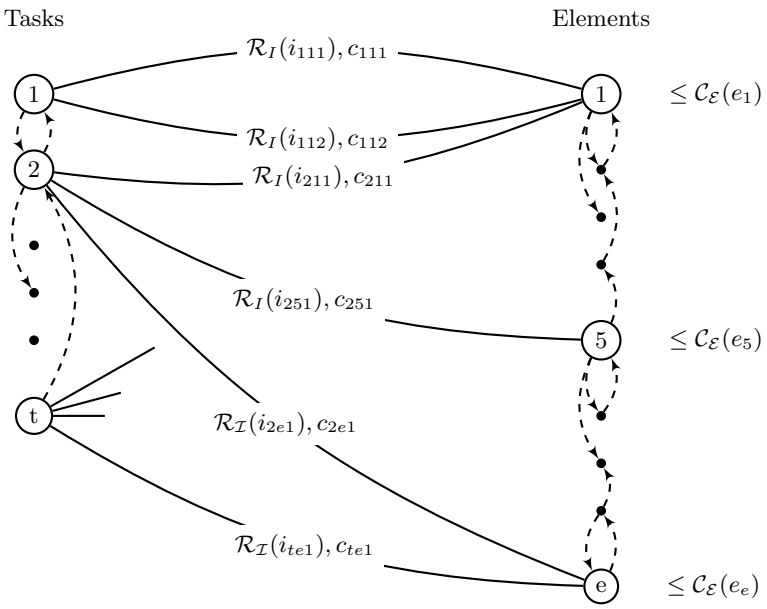


Figure 2.2: A graphical representation of the multilevel generalized assignment problem solved in the binding phase

elements is chosen. This problem is an instance of the Multilevel Generalized Assignment Problem (MGAP) and can be mathematically formulated as follows:

$$\text{minimize} \quad \sum_{t\in\mathcal{T}}\sum_{e\in\mathcal{E}}\sum_{i\in\mathcal{I}(t)} c_{tei}x_{tei} \tag{2.2.1}$$

$$\text{subject to} \quad \sum_{e\in\mathcal{E}}\sum_{i\in\mathcal{I}(t)} x_{tei} = 1 \quad \forall t \in \mathcal{T}, \tag{2.2.2}$$

$$\sum_{e\in\mathcal{E}}\sum_{i\in\mathcal{I}(t)} R_\mathcal{I}(i)x_{tei} \leq C_\mathcal{E}(e) \quad \forall t \in \mathcal{T}, \tag{2.2.3}$$

$$x_{tei} \in \{0,1\} \quad \forall t \in \mathcal{T}, \forall e \in \mathcal{E}, \forall i \in \mathcal{I}(t)$$

In this mathematical model, assignment of task implementations to elements is modelled by the variable x: x_{tei} is set to 1 if and only if implementation i of task t is mapped to element e. Equation (2.2.2) states that exactly one implementation of each task must be mapped to a processing element. Equation (2.2.1) models the *cost* of mapping implementation i of task t onto element e with c_{tei}. Finally, equation (2.2.3) states the constraint that per processing element the total amount of resources used is not higher than the element's capacities. Note that no topological constraints such as availability of enough bandwidth between communicating tasks are taken into account during this phase. These constraints are dealt with in subsequent phases.

A polynomial-time algorithm that provides approximate solutions to a GAP is given in [18] and uses a measure of desirability of assigning an item j to knapsack i. All unassigned items are iteratively considered and the item j having the maximum difference between the largest and second largest desirability is assigned to the knapsack for which the desirability is maximal.

2.2.2 Mapping

During the mapping phase the task implementations that were selected in the binding phase are mapped to specific elements in the hardware platform. Mathematically this may again be modelled as an integer linear program, which is similar to the program given by equations (2.2.1), (2.2.2) and (2.2.3) with the choice of task implementation fixed:

$$\text{minimize} \quad \sum_{t \in \mathcal{T}} \sum_{e \in \mathcal{E}} c_{te} x_{te} \tag{2.2.4}$$

$$\text{subject to} \quad \sum_{e \in \mathcal{E}} x_{te} \leq 1 \quad \forall t \in \mathcal{T} \tag{2.2.5}$$

$$\sum_{e \in \mathcal{E}} R_{\mathcal{J}}(\mathcal{J}_t) x_{te} \leq C_{\mathcal{E}}(e) \quad \forall t \in \mathcal{T}, \tag{2.2.6}$$

$$x_{ei} \in \{0, 1\} \quad \forall t \in \mathcal{T}, \forall e \in \mathcal{E}$$

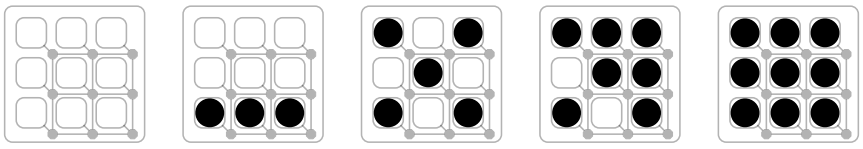
Each element chooses a set of tasks by using a cost function. The goal of this choice is to minimise the costs of mapping the chosen tasks to that element while satisfying the capacity constraint for that element. The cost c_{te} of mapping task t to element e takes into account the spatial properties of the platform. The approach taken in the mapping phase aims at keeping tasks close to their adjacent tasks in the task graph while at the same time avoiding fragmentation of the hardware platform. This is achieved by mapping the application's tasks iteratively in a breadth-first fashion.

Given a task t in the task graph, the subset of tasks $\mathcal{T}_i$ is the set of tasks that are at depth i of the task graph's breadth-first search tree rooted at t. The mapping algorithm iterates over the iso-distant tasks $\mathcal{T}_i$. After determining $\mathcal{T}_i$, a set of elements $\mathcal{E}_i$ is then collected such that for each task in $\mathcal{T}_i$ at least one element is available. Elements are chosen as close as possible to the elements selected in the previous iteration, $\mathcal{E}_{i-1}$. The task of finding an optimal mapping $\mathcal{M}_i$ of each task in $\mathcal{T}_i$ to an element in $\mathcal{E}_i$, taking into account resource constraints, is an instance of a generalized assignment problem and is 'solved' heuristically using the approach from [4].

The following sections give an overview of the task-to-element costs used in equation (2.2.5).

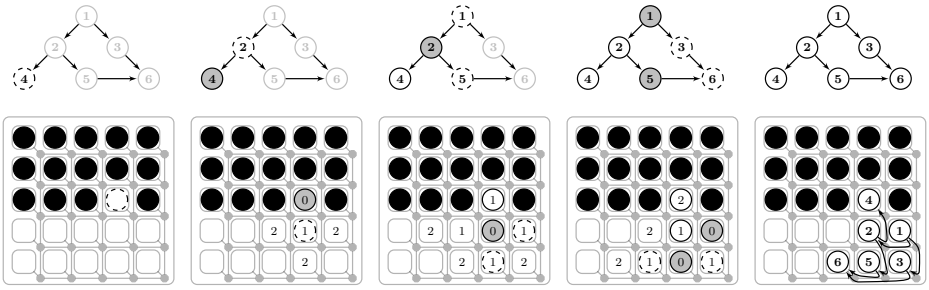
2.2.2.1 External fragmentation

External fragmentation f_{ext} is defined as the ratio between the number of adjacent element pairs of which only one element is available and the total number of adjacent elements in the system. With this definition, the external fragmentation of an empty or fully utilised platform is 0%. External fragmentation is maximal for a checkerboard pattern, see figure 2.4.



(a) $f_{ext} = 0\%$ (b) $f_{ext} = 25\%$ (c) $f_{ext} = 100\%$ (d) $f_{ext} = 50\%$ (e) $f_{ext} = 0\%$

Figure 2.3: External fragmentation on some example platforms.



(a) $i = 0$ (b) $i = 1$ (c) $i = 2$ (d) $i = 3$ (e) finished

Figure 2.4: Mapping state after each iteration in *MapApplication*, where gray nodes represent $\mathcal{T}_{i-1}$ and dashed nodes $\mathcal{T}_i$. On the bottom row the final search state of *SearchElements* is given.

The external fragmentation heuristic makes better connected elements more costly to use. The cost of external fragmentation is reduced when elements close to an element e are used by tasks that are close to the task t mapped to element e.

2.2.2.2 Communication distance

While searching the platform for elements $\mathcal{E}_i$ to map tasks onto, the system keeps track of the distance between newly discovered elements and their predecessors in $\mathcal{E}_{i-1}$. Tasks that are close in the task graph and thus are mapped during subsequent iterations to elements in $\mathcal{E}_i$ and $\mathcal{E}_{i-1}$ should ideally be maped onto elements that are close on the platform. Therefore, a heuristic for the cost of mapping task t to element e is obtained by summing the distances between e and elements e' onto which neighbourhood tasks are mapped.

2.2.3 Routing

During the routing phase each channel that connects tasks t_i and t_j in the SDF graph is mapped to a route between the network interfaces of the processing elements hosting t_i and t_j. This mapping is again a search problem as not all routes are possible and the goal is to find the least expensive route between two endpoints. The routing algorithm used by Kairos is *Uniform Cost Search* (UCS, [21]), which employs Dijkstra's algorithm [7] but terminates when the destination is reached. This approach is similar to the approach described in [19].

2.2.4 Validation

After an execution layout has been found for the application, the application's actual performance must meet the stated performance constraints (throughput, latency and / or jitter). As the application's performance may not straightforwardly be determined from the recorded performance information of isolated tasks, this information is collected at runtime.

Worst-case performance analysis may be used to decide whether the application will meet its performance constraints, but overly pessimistic worst case assumptions may lead to false rejection of the found execution layout.

Rather than making a poorly founded decision based on statically known performance data, the ultimate goal of ADVANCE is to observe the application's performance using the chosen mapping and learn which mapping may improve the performance using the observed data.

2.3 Heuristic feedback

The presented workflow is a *hierarchical search with iterative refinement*. In case the execution layout that was found during the binding, mapping and routing phases is found to meet the performance requirements, the process terminates with an outcome of success. When during the validation phase it is found that performance requirements are not met, no information is tracked back into the higher levels of the workflow. As a result, the space of possible execution layouts is traversed in a possibly inefficient way. Consider the case where a task t is the last task to be mapped during the binding and mapping phases. If, during validation it is found that task t is part of a *critical cycle* in the system, task t should be assigned a higher priority and be mapped before less critical tasks are mapped.

In order to exploit information obtained during the validation phase, first of all criticality-related information must be made available during the validation phase. This involves, apart from determining the throughpt of the mapped application (modelled as an SDF graph), determining the critical cycle in the corresponding HSDF graph (in terms of max-plus algebra this is analogous to finding a basis of the system's eigenspace).

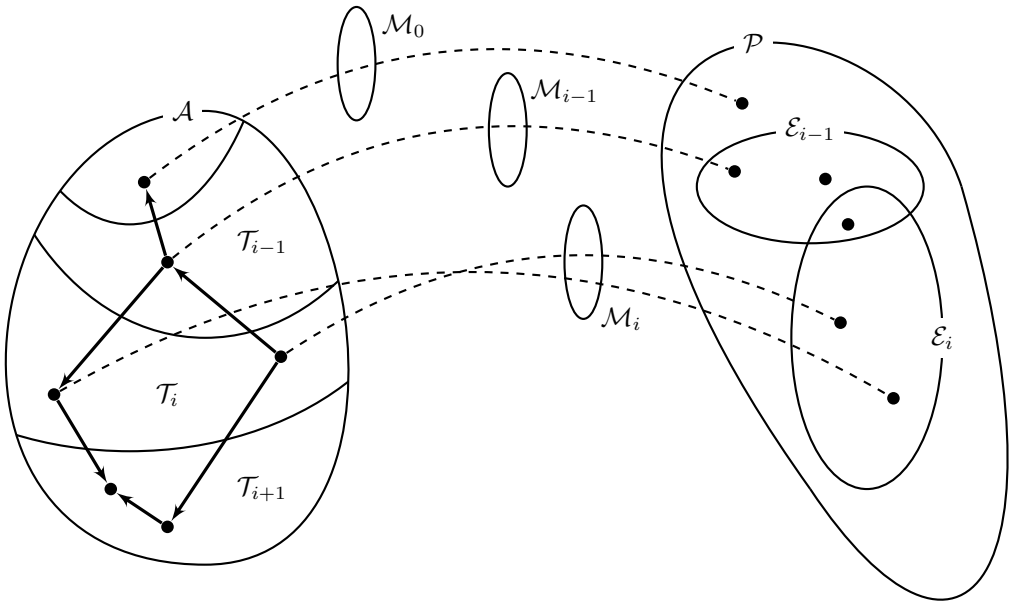


Figure 2.5: For every subset $\mathcal{T}_i$ of tasks in the application a subset $\mathcal{E}_i$ of the platform's elements is selected to form mapping $\mathcal{M}_i$

Secondly, heuristics for the costs of mapping a task to an element should include the task's criticality.

2.3.1 Finding critical tasks

In an SDF graph, some actors have no impact on the graph's throughput - their scheduled execution may be postponed (to a certain extent) without affecting the remaining schedule. Other actors are *critical* - delaying some (or all) of their executions subsequently delays other executions and as a consequence decreases the throughput of the entire graph.

Determining which actors in an SDF graph is critical involves finding the *critical cycle* in the corresponding HSDF graph [22], [20]. State-of-the-art methods to do this are by simulating the SDF graph until a periodic phase is found [10] or by applying an algorithm to find the *maximum-cycle-ratio* of the HSDF graph [6], [5]. In terms of computational efforts, both methods have their disadvantages. The simulation approach is sensitive to differences between the execution times of individual actors whereas the HSDF-based approach is completely insensitive to execution times but rather dependent on the size of the HSDF graph (which may be exponential in the size of the original SDF graph, in the worst case, [22]).

In this section we sketch a novel algorithm that may be used to identify the critical cycle in an HSDF graph. We furthermore quantitatively compare this new

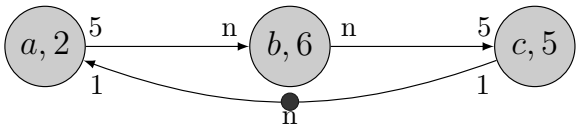


Figure 2.6: An SDF cycle for which the size of the (timed) marked graph grows linearly as n increases, whereas the size of the explored state-space remains constant. The maximum cycle ratio of the corresponding timed marked graph is 13, independent of n.

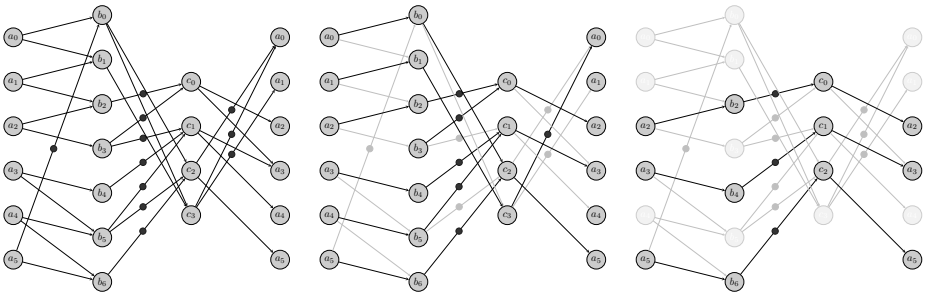
algorithm with both state-of-the-art approaches.

2.3.1.1 Fast SDF cycle analysis

The simplest SDF graph that has a maximum achievable throughput is an SDF cycle [11]. When following the state-of-the-art approaches for analysing such a cycle, then both simulation and HSDF-based analysis have their disadvantages. Using simulation to find the throughput of an SDF graph is analoguous to the power algorithm and thus is sensitive to the execution times of the actors in the SDF graph [10]. An HSDF-based analysis on the other hand, may require the construction of a very large HSDF graph, which is both time and space consuming [10, 22]. Inefficiency of the HSDF-based analysis is illustrated by the SDF graph shown in figure 2.6: The state-space exploration based approach requires, independent of n, exactly 4 steps, whereas the size of the HSDF graph increases linearly with n.

An HSDF graph corresponding to a multirate SDF graph contains many redundant edges. This redundancy becomes apparent when representing edges in an HSDF graph as a set of constraints in max-plus algebra [2, 3, 10]. Removing these redundant edges greatly simplifies the HSDF graph and its analysis. In case the SDF graph is a single cycle, removing all redundant edges from the corresponding HDSF graph reduces the indegree of each HSDF actor to one. As a result, the HSDF graph is reduced to a set of components each of which contains precisely one circuit (such a graph is called a sunflower bouqet graph in [14]). Figure 2.7 shows how the HSDF graph corresponding to an SDF cycle changes as redundant edges and actors are removed changes into a graph consisting of three critical cycles.

Now an important result is that each of these cycles has both the same weight as the same number of tokens. This is due to the fact a property of an HSDF graph is that an edge *crossing* a similar (i.e., represents the same SDF channel) edge crosses all edges that are parallel to the edge crossed. Figure 2.8 shows two graphs that seem to be counterexamples to the claim, but these graphs are illegal as they violate the stated property of edges in an HSDF graph. As a result, each cycle in the edge-reduced HSDF graph is equally critical. This property may be exploited in throughput analysis of SDF cycles and leads to a very efficient algorithm that is based on only a *partial* construction of the HSDF graph, avoiding the potentially huge size of the HSDF graph. This algorithm is listed in algorithm 1.



(a) original hsdf graph

(b) graph after removing redundant channels

(c) final graph

Figure 2.7: Overview of the reduction of the HSDF graph corresponding to the SDF cycle shown in figure 2.7. After removing redundant channels and actors, the final graph contains three cycles, all of which are critical.

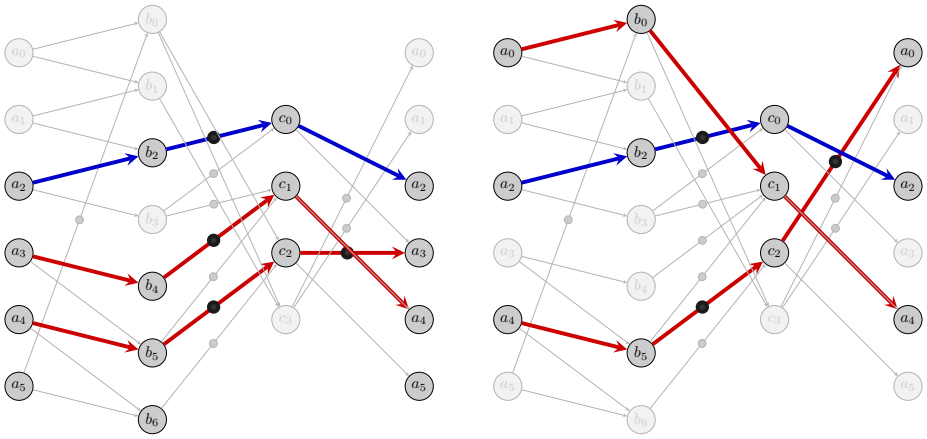


Figure 2.8: Two impossible HSDF graphs, both violating the property that an edge crosses all similar edges carrying less tokens.

input : A consistent SDF cycle $\mathcal{G}$ with
- channels $\mathcal{C} = \{c_0 = (a_0, a_1), \ldots, c_{n-1} = (a_{n-1}, a_0)\}$
- actors $\mathcal{A} = \{a_0, \ldots, a_{n-1}\}$
- repetition vector q

output : The critical cycle of the HSDF graph corresponding to $\mathcal{G}$

$c = (a_j, a_i) \leftarrow$ arbitrary channel in $\mathcal{C}$;
$k \leftarrow \left\lfloor \frac{\texttt{tokens}(c_m)}{\texttt{cons.rate}(a_i)} \right\rfloor$;
$P \leftarrow \langle\rangle$;
while a_i^k *not on path* P **do**
  append a_i^k to P;
  $j \leftarrow j + 1 \bmod n$;
  $i \leftarrow i + 1 \bmod n$;
  $c \leftarrow (a_j, a_i)$;
  $k \leftarrow \left\lfloor \frac{k \cdot \texttt{prod.rate}(a_j) + \texttt{tokens}(c_m)}{\texttt{cons.rate}(a_i) \cdot q_i} \right\rfloor$;
end
return `dropwhile(`$\neq a_i^k$`) P`

Algorithm 1: Algorithm to obtain the critical cycle in a homogeneous SDF graph corresponding to a consistent multirate SDF graph. The algorithm constructs only a small part of the entire HSDF graph (which may be very large)

	Average	Std. Dev
SDF actors	$3.0 \cdot 10^1$	$1.2 \cdot 10^1$
HSDF actors	$7.6 \cdot 10^3$	$3.1 \cdot 10^3$
Initial tokens	$9.4 \cdot 10^5$	$4.4 \cdot 10^5$
$\min_a q_a$	$2.9 \cdot 10^1$	$2.2 \cdot 10^1$
Crit. cycle len	$2.4 \cdot 10^2$	$1.8 \cdot 10^2$

(a) Testset details

Set		Min	Max	Avg	sd
A,B	Analysis	10	1613	$3.0 \cdot 10^2$	$1.9 \cdot 10^2$
A	Simulation	18	1673	$4.2 \cdot 10^2$	$2.2 \cdot 10^2$
	Ratio (S/A)	1.0	48.6	1.7	1.1
B	Simulation	132	59616	$1.0 \cdot 10^4$	$7.4 \cdot 10^3$
	Ratio (S/A)	6.0	702.2	$3.7 \cdot 10^1$	$2.8 \cdot 10^1$

(b) Analysis vs. Simulation

Table 2.1: Number of steps required by analysis and simulation on a testset containing 50,000 random SDF cycles. The number of simulation steps depends on the execution times of the SDF actors, as shown.

We have applied our novel throughput analysis of SDF cycles to a set of 50,000 randomly generated SDF cycles, the properties of which are shown in table 2.1a. For comparison, we have applied the simulation approach to the same set of cycles. Rather than comparing CPU time of both algorithms we have compared the number of *steps* required. A single step in simulation involves simulating the effects of completed actor firings (producing tokens on outgoing channels and possibly starting new firings) and updating the remaining firing times of executing actors. A step in our approach is a single iteration of the *while*-loop shown in algorithm 1. Because the number of steps required by simulation is dependent on the differences between actor execution times, we have created two versions of the test set: In set A each actor has the same execution time, whereas in set B the execution time of each actor is drawn from a uniform distribution. Table 2.1 shows the statistics for the required number of steps by simulation and our approach (labelled *analysis*). Note that results for analysis are independent for the test set used. We have included a *ratio* statistic which is calculated for each SDF cycle in the test set by dividing the number of steps required by simulation by the number of steps required by our approach. Table 2.1 shows that when execution times are different (set B), simulation needs on average 37 times as many steps as our approach, and more than 700 times as many steps in the most extreme case. It appears that the described HDSF-based approach for finding the throughput of an SDF cycle requires, in the worst case, the same number of steps as simulation does. Generally the described method outperforms simulation, especially when the actor execution times differ.

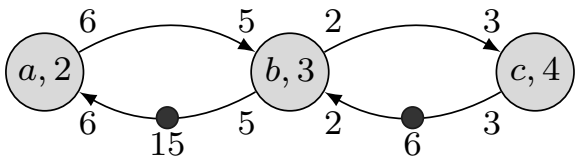


Figure 2.9: An SDF graph consisting of two (simple) cycles. The maximum cycle ratios for the left and right cycle are 15 and 14, respectively. The entire graph as a whole has a maximum cycle ratio of 19.

2.3.1.2 Analysing arbitrary SDF graphs

The maximum achievable throughput of a *homogeneous* SDF graph consisting of multiple cycles is the minimum of each cycle's maximum achievable throughput. For a multirate SDF graph, this is different. Consider the example in figure 2.9. In this figure, the critical cycle in the HSDF graph is not entirely contained within either of the two simple cycles in the SDF graph. Figure 2.9 shows that the maximum achieveable throughput of two cycles do not trivally compose.

In order to find the throughput of an arbitrary SDF graph we may start by using the algorithm described in the previous section to find the throughput of one of the graph's simple cycles. We may then incrementally add edges to the SDF graph and update the critical cycle in the corresponding HSDF graph as new cycles are formed. This process stops as soon as the entire SDF graph has been fully reconstructed, yielding the critical cycle of the corresponding HSDF graph, from which the graph's throughput may easily be derived. Further work is required to evaluate this compositional approach to timing analysis of SDF graphs and compare it to existing state-of-the-art methods.

2.4 Discussion

Mapping a stream processing application to a multiprocessor system under performance constraints is a complex problem. Heuristics are used to effectively prune the search-space of possible mappings in order to find a near-optimal solution fast. These heuristics take into account the communication costs between tasks and the fragmentation of the hardware platform.

Since information regarding performance is not available until the application has been fully mapped, this information can not be included in the costs used in the search for a viable mapping. However, once the mapped application has been tested for its performance, tasks and communication channels may be ordered according to their impact on the mapped application's performance.

In this chapter we have presented a novel approach to the analysis of SDF cycles which, faster than known state-of-the-art methods, yields the maximum achievable throughput of the SDF cycle. This analysis may be used during the

mapping workflow to perform early validations once each cycle in the SDF graph has been layed out. In case the resulting performance does not meet the constraints, rather than remapping the entire application the resource manager may prioritize the most critical task (the task having most impact on the application's performance).

The current implementation of the resource manager deals with applications that may be modelled as synchronous dataflow graphs. Although task execution times and communication latencies may vary during an actual execution of such an application on a hardware platform, the structure and schedules may be determined at compile-time. Heuristics that work for worst-case cost and resource usage estimates need to be extended to deal with execution time variations and dependent execution times.

The applications targeted by the ADVANCE project are more dynamic in nature. Apart from variations in box execution times due to data-dependent behaviour, a box in an S-Net may send arbitrary numbers of records with various types on its output channels, which may lead to non-deterministic control flow. In order to take into account these many possible flows through an S-Net, heuristics need to incorporate the activity of boxes and channels. Communication distance and resource fragmentation may be extended to probabilistic measures using the resource usage models developed in work packages 3 and 4. The next chapter elaborates further on this.

Chapter 3

Exploiting resource usage information

This chapter explores how resource usage information given in the form of statistical profiles may be used heuristically in the mapping problem for S-Net applications. As these statistical profiles are not yet available, rather than referring to existing profiles we state a number of assumptions about these profiles that need to be fulfilled.

Within this project the main challenge of work package 6 is to extend the current resource management approach, which is based on synchronous dataflow models, to allow it to deal with a more dynamic class of applications that may be modelled as S-Nets. Therefore, this chapter addresses the main differences between S-Nets and (synchronous) dataflow graphs and how they lead to new heuristics to be used in the mapping process.

As is pointed out in the previous chapter, mapping an application modelled as an SDF graph to a hardware platform such that performance constraints are met is a hard task, even when resource usage information is rather simple (i.e., fixed worst-case execution times expressing the usage of cpu time). Even when moving to a statistic description of resource usage for SDF graphs, resource usage information for SDF graphs remains relatively straightforward: as the flow of data through the graph is fixed the *order* of execution (the task schedule) is known in advance. For S-Nets, resource usage becomes more dynamic: a box in an S-Net may produce records of different types, which may lead to records following different routes through the S-Net [12]. We use *record flow* to indicate the route a record follows in S-Net. Dynamicity of record flow is a key difference between SDF and S-Nets and poses a challenge for resource management.

When execution times vary the complexity of the problem of finding a near-optimal mapping increases:

3.1 From synchronous dataflow to S-Net

The resource management system Kairos currently developed within the Computer Architectures for Embedded Systems group at the University of Twente is targeted at stream processing applications. These applications are modelled as synchronous dataflow (SDF, [17]) graphs. Synchronous dataflow models offer many advantages to the domain of firm real-time embedded systems: Using worst-case estimates for actor execution times, an SDF graph may be analysed for performance characteristics such as maximum throughput and minimum latency. This allows system designers to design a system such that it meets strict performance requirements such as minimum throughput and / or maximum latency.

The analysibility of SDF models comes at the cost of poor expressiveness. Although many stream processing applications may be elegantly modelled by an SDF graph, SDF models are incapable of expressing dynamic aspects of modern streaming applications. The dynamism in modern streaming applications often originates from different *modes of operation* in which data production and / or execution times may differ [24].

When it comes to expressiveness, S-Nets are far richer than synchronous dataflow graphs: S-Nets may be translated into the functional programming language Haskell [15] and thus are *turing complete*, and therefore may be used to model any computer program. This rich expressiveness makes many performance questions that are relatively easy for SDF models, generally undecidable.

Within work package 6 of the ADVANCE project, the goal is to use statistical information regarding the observed dynamics of an S-Net in order to make better resource management decisions. In terms of resource management the main difference between a *data streaming* application that may be modelled as an SDF graph and a more general class of applications that may be modelled as an S-Net is that in S-Net the arrival or activation order and execution time of *boxes* is not known a priori. This uncertainty requires new clever heuristics based on observed statistics, which may be inspired by the heuristics used for SDF models.

The following sections propose a number of heuristics.

W

3.2 Heuristics for resource management with uncertainty

We regard resource management as an *online* task: as an S-Net is executed, boxes become activated (when an incoming record is available) and must be placed on a processing element in the hardware platform. Placements should be made with the goal to optimise for various performance metrics such as minimum throughput or maximum latency and / or jitter. Although we have no knowledge regarding future box execution times and record flow, statistical information is collected over the past and may be used to predict to some extent the future behaviour.

Resource usage profiles are developed within work package 4. Since no resource

usage profiles are available as of yet, we rely on a few basic assumptions concerning the information stored in these profiles.

3.2.1 Statistical resource usage profiles

As stated before, no resource usage profiles are available as of yet. The goal of these resource usage profiles is to capture the non-determinism inherent to an S-Net application mapped to a hardware platform and provide some form of model that has predictive power. As such, the minimal assumptions regarding these usage profiles are the following:

- A conditional probability table for each box, listing the probability that that box will produce a record of type A after consuming a record of type B. Note that a box always consumes a single record but may produce arbitrary numbers of records (output record types are predetermined).

- A conditional cumulative density function that relates the latency of a box to its cumulative probability given the type of the consumed input record.

This assumed usage information may be used to extend the heuristics that are currently in use by the resource manager. This is described in the following section.

3.2.2 Extending SDF-based heuristics

The heuristics currently used by the resource manager are based on applications that may be modelled by SDF graphs. A great advantage of an SDF model compared to S-Net is that in an SDF graph the flow of data is fully deterministic and predetermined. This deterministic nature first of all makes determining the inter-task communication costs, once a mapping is found, straightforward. For S-Net however, things are a bit different. The two heuristics described in the previous chapter (external fragmentation and communication cost) become stochastic variables once they are applied to S-Net networks.

Whereas communication is fixed in SDF, it is not in S-Net since the record flow in S-Net networks is non-deterministic [13]. The amount of external fragmenation is a measure on the scattering of free resources among the resource used to run tasks. When statically mapping a box to a processing element (i.e., the box runs on the same processing element for the duration of the application), the resources it uses vary over time. Box execution behaviour may vary from strongly periodic to highly sporadic. It is generally not possible to derive a box's execution behaviour from the S-Net model. External fragmenation is therefore probabilistic as well.

When two tasks t_1 and t_2 are mapped to two different processing elements e_1 and e_2, the cost of communication is simply the length of the shortest available path between e_1 and e_2. In order to cope with the non-deterministic nature of S-Nets, we simply need to extend communication cost into a probability mass function. The probability of a communication path being used can be derived from

the statistical resource usage profiles. To do so, we model the stream through a network as a Markov model, using the observed statistical resource usage profiles as state transition probabilities. The stationary distribution of the markov chain maps each box in the network to the probability that at a random point in time this box will be processing an input record.

3.3 Conclusion

The heuristics used in the mapping of an application modelled by an SDF graph to a hardware platform, are unable to deal with the non-determinism found in S-Net networks. To solve this problem, we may model the record flow through an S-Net network by a markov chain, where each state maps to an S-Net box. This approach leads to heuristics that are based on the expected value of communication cost and external fragmentation. In order to evaluate these heuristics applications showing highly dynamic behaviour are needed. Since only a limited number of realistic applications is available, the following chapter describes the construction of a test set of applications that may be used to benchmark the heuristics described in this chapter.

Chapter 4

Evaluation

In this chapter we describe a work plan to evaluate the heuristics detailed in the previous chapter. As within the ADVANCE project realistic industrial applications are not due until year 3 we have chosen to compile a set of applications that may be used to benchmark our heuristics. A strong benefit of such a test set is that subtle changes in existing heuristics or entirely new heuristics may be easily evaluated and quantitatively compared.

A difficulty in generating realistic benchmarks is first of all due to the rich expressiveness of S-Nets. As the S-Net model is turing complete it may model any computation, including those that would lead to an unbounded usage of memory or computation resources. It is clear that for unbounded computations resource management is rather useless as resources are limited by definition. In order to restrict the generated test instances to those that have bounded resource usage we choose an application model that is slightly less expressive than S-Net.

4.1 Generating random S-Nets

An S-Net box specifies the types of input records and output records. Each box consumes exactly one record, but may produce a finite number of output records. As a result, due to synchronisation it may occur that records are pending in the system waiting to be synchronised by another record. Because boxes in S-Net are black boxes in terms of their behaviour, properties such as deadlock-freeness and boundedness are undecidable. It is the responsibility of the system designer to ensure that the number of produced records are balanced in such a way that neither resource deadlock nor starvation will occur.

When randomly generating S-Nets and box behaviours, it is impossible to guarantee boundedness and deadlock-freeness when treating the boxes as true black boxes. Failing to explicitly take into account the maximum amount of records available in the systems will result in an extremely low likelihood of generating S-Nets that may run infinitely.

We solve this problem by limiting the expressiveness of S-Net somewhat and

move to a *scenario-aware dataflow* (SADF, [24], [9]) approach. We believe much is to be gained from choosing a somewhat easier model before moving to a more complex one.

4.2 S-Net and scenario-aware dataflow

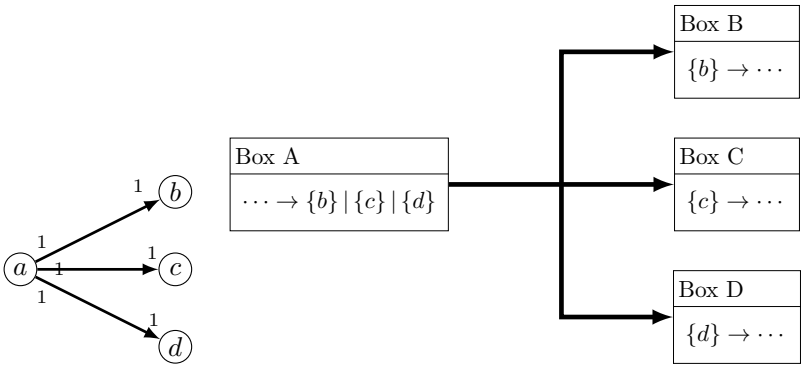
Scenarios describe how a system is used. Where use-case scenarios describe how a system is used from a user's perspective, *application scenarios* characterise a system's use from a resource usage perspective. As we look at systems from a resource usage perspective, we will use the term "scenario" to refer to application scenarios. Different scenarios may involve different numbers of tasks involved in a computation, such as different recursion or iteration depths for recursive algorithms. Scenarios may also distinguish between typical execution times of tasks, such as for example in a video decoder for MPEG video where intra frames typically require more processing time than predicted frames. Another example of different scenarios is found in the X-ray image processing use case from WP7, where the object being tracked sits still or is moving.

Scenarios may be derived from the source code of the application, which in the finest form may result in a separate scenario for each possible path through the application's code. Another way to separate out several scenarios is by taking a data perspective. Scenarios may be constructed for different modes of the input data. This is sensible when many of the tasks (such as in video decoding or feature tracking) have highly data-dependent behaviour. As such, scenarios may remove some of the correlation beteen task execution times caused by co-varying data-dependent behaviour.

The idea of extending synchronous dataflow with scenarios has resulted in several new models in which production and consumption rates of actors change over time, such as cyclo-static dataflow [1] or variable rate dataflow [25]. Although these models are more expressive than the basic synchronous dataflow model, the scenarios allowed by these models are repetitive by nature. As a result, these models do not allow arbitrary, non-deterministic orderings of scenarios.

The *Scenario-aware dataflow* (SADF, [24, 9]) model extends the synchronous dataflow model with scenarios. The model is similar to hybrid models ([8], [16]) where synchronous dataflow is embedded in finite state machines. As transitions between scenarios are non-deterministic, SADF suits applications where the modes or features of the data are unpredictable but where the data-dependent behaviour is predictable.

S-Net allows non-deterministic behaviour by nature. A box produces records of which the possible types are known at compile-time, but the number and type of records created at runtime is fully dependent on the box implementation. As such, a box may produce a different number of records of different types each time the box is executed. Furthermore, the flow of records through an S-Net network is non-deterministic by nature. The network combinator *parallel composition* models



(a) SDF (b) S-Net

Figure 4.1: An actor having multiple outgoing channels in SDF is modelled in S-Net by a box that produces records with distinct types, one type for each predecessor.

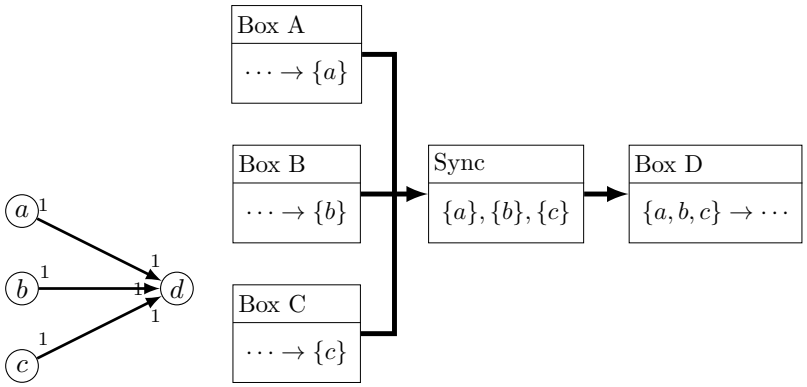
non-deterministic choice - if a record's type matches multiple type signatures it depends on the scheduling technique to which of the boxes the record is sent to.

The main difference between an S-Net and a scenario-aware dataflow model in terms of how non-determinism is treated is that in an SADF model non-determinism is only found in the transitions between scenarios. Each scenario in an SADF graph is modelled by a single SDF graph. An SDF graph may be translated in a straightforward way into an S-Net network. This translation is described in the following section, after which the encoding of scenarios within S-Net is detailed.

4.3 Translating SADF into S-Net

There are many differences between synchronous dataflow graphs and S-Net networks. As the name already states, in SDF graphs synchronisation is implicit. In S-Net networks on the other hand, communication is asynchronous. Should communication be synchronous then this needs to be modelled explicitly in an S-Net network.

Another difference lies in the fact that within S-Net, boxes are black boxes that are typed but otherwise unspecified. An actor in an SDF graph specifies the amount of data processed by consumption and production rates, without specifying the types or values of the tokens. Actors in an SDF graph may have an arbitrary number of outgoing channels, whereas boxes in S-Net are single-in single-out. An SDF actor with multiple outgoing channels may be modelled in S-Net by encoding the destination of a record in the type of that record (see figure 4.1).



(a) SDF (b) S-Net

Figure 4.2: An actor having multiple incoming channels in SDF is modelled in S-Net by synchronising the records produced by successors boxes.

4.3.1 Synchronous dataflow in S-Net

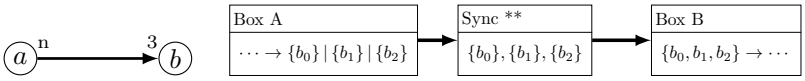
We can model actors with multiple outgoing channels in S-Net by using a parallel split and ensure deterministic record flow by matching the types of produced records by the input record type of successor boxes. This is shown in figure 4.1. Note that the *number* of records produced is not specified; a box may produce any number of each of the possible output record types.

Synchronisation takes place in two different ways in a synchronous dataflow graph. First (*channel synchronisation*), an actor with multiple incoming channels synchronises its data consumption with the production of data of each of the actor's predecessors. Second (*token synchronisation*), an actor's consumption rate specify that each execution of that actor is synchronised with the reception of a number of tokens. Synchronisation has to be made explicit in S-Net by a synchrocell. In both cases synchronisation is continuous, which means the synchrocell is wrapped in a star combinator.

Channel synchronisation may be modelled in S-Net by associating each of the outgoing channels of an actor in the SADF graph with a different record type, which matches the output signature of the source of that channel. This ensures that record flow through the net is deterministic. Figure 4.2 shows an example of an S-Net network modelling channel synchronisation.

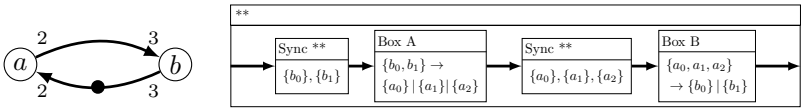
Token synchronisation is modelled in S-Net by replacing a channel with two streams connected by a synchrocell. The number of slots in the synchrocell is equal to the channel's consumption rate and the type of each slot equals the type of records produced onto that channel. This is illustrated in figure 4.3. Note that initial tokens are not modelled explicitly in S-Net; an SDF graph that is deadlocked may still be modelled in S-Net.

Finally, networks in S-Net are acyclic whereas dataflow graphs are cyclic.



(a) SDF (b) S-Net

Figure 4.3: Synchronisation due to differences in production and consumption rates. This synchronisation is implicit in SDF but needs to be modelled explicitly in S-Net.



(a) SDF (b) S-Net

Figure 4.4: S-Net networks are acyclic, whereas in SDF cycles occur naturally. An SDF cycle, if deadlock free, may be unfolded an infinite number of times. In a similar fashion, an S-Net hides cycles by allowing infinite graphs. Therefore SDF cycles may be transformed into S-Net networks as shown.

Cycles in dataflow graphs model infinite applications of functions to an infinite input stream. In other words, dataflow graphs use cycles to hide the infinite nature of the graph. In a similar fashion, S-Net networks may represent computations that are run on an infinite stream of input data. Rather than using cycles to hide infinity, S-Net uses the *serial replication* combinator ("*") to denote a serially composed, infinitely repeating network. Translating SDF cycles into S-Net is therefore a straighforward application of the serial replication combinator and is illustrated in figure 4.4.

4.4 Scenarios in S-Net

Within a scenario in SADF the flow of data is deterministic and actor execution times are independent. This translates to a deterministic flow of records through the network in S-Net. Note that the assumption of independent execution times may not be realistic as complex interactions between software processes and hardware elements (e.g., bus arbitration, cache effects) will most likely result in some correlation, even when all correlation due to data dependent behaviour has been removed.

In order to ensure a deterministic record flow we must ensure that boxes produce records in a deterministic way, i.e. a fixed number of records of each of the types specified in the box output signature. This may be achieved by encoding the current scenario in the values contained in the records sent through the network. A box may then, for example by using a lookup table, read its output requirements indirectly from the active scenario. Although this will ensure deterministic record flow through the network, it is harder to control the execution time of each box. An easy but unrealistic approach is to have each box simply wait for an amount

of time, determined by a scenario-dependent probability distribution. Such an approach would result in (close to) independent execution times but unrealistic as it would hide the complex interactions between software and hardware that make performance prediction so hard.

4.4.1 Stochastic scenario transitions

When record flow for a given scenario is fully deterministic, non-deterministic record flow is captured by non-determinism in the order of scenarios. There are many ways to achieve this. In the most extreme case, scenarios may be chosen in a purely random way. Because different scenarios typically represent different modes of the data, such an extreme way of non-determinism may not be very realistic. We therefore choose to model the non-deterministic sequence of scenarios by a *markov process*.

4.4.2 Non-deterministic execution times

In order to achieve non-deterministic behaviour with regard to execution times of boxes we may simulate a computation. There are several ways to make the time of this computation behave non-deterministically. A straightforward way is to let the computation consist of a number of iterations (e.g., in a *for*-loop) and draw this number from some parametric probability distribution. In order to include complex effects due to bus arbitration, cache effects, interruption by kernel tasks, etc., the computation should consist of communication with (shared) memory.

4.5 Future work

This chapter has described the process of generating random applications, modelled as S-Nets. These applications may be used as a benchmark for the resource manager developed in the Advance project. Heuristics derived from statistical resource usage profiles, which are under development in WP3, may be evaluated using the benchmark set.

Furthermore, we believe that actually running the generated applications on a heterogeneous multiprocessor system may provide a fertile ground for work packages involved with monitoring the performance of mapped applications (WP3, WP6) and the development of new statistical models aimed at composability (WP4). It is expected that close cooperation with these work packages will yield numerous ideas concerning the parameters used to generate random S-Nets.

Randomly generated applications provide unlimited use cases to the resource manager and may be used to compare and improve heuristics. However, it is difficult to project any conclusions drawn from random use cases regarding the effectiveness of heuristics onto the realistic applications developed in WP7 of this project. A final and concluding evaluation of heuristics can therefore not be made before the final year of the Advance project.

Bibliography

[1] G. Bilsen, M. Engels, R. Lauwereins, and J.A. Peperstraete. *Cyclo-static data flow*. IEEE.

[2] G Cohen. Max-plus algebra and system theory: Where we are and where to go now. *Annual Reviews in Control*, 23:207–219, 1999.

[3] Guy Cohen, Geert Jan Olsder, and Jean-pierre Quadrat. *Synchronization and linearity*. Wiley New York, 1992.

[4] R Cohen, L Katzur, and D Raz. An efficient approximation for the Generalized Assignment Problem. *Information Processing Letters*, 100(4):162–166, November 2006.

[5] Ali Dasdan. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 9(4):385, 2004.

[6] Ali Dasdan, Sandy S. Irani, and Rajesh K. Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. *Annual ACM IEEE Design Automation Conference*, page 37, 1999.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[8] G Gao. An Efficient Hybrid Dataflow Architecture Model. *Journal of Parallel and Distributed Computing*, 19(4):293–307, December 1993.

[9] Marc Geilen. Synchronous dataflow scenarios. *ACM Transactions on Embedded Computing Systems*, 10(2):1–31, December 2010.

[10] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij. Throughput Analysis of Synchronous Data Flow Graphs. *ACSD*, 2006.

[11] A.H. Ghamarian, M. Geilen, T. Basten, B. Theelen, M.R. Mousavi, and S. Stuijk. Liveness and boundedness of synchronous data flow graphs. *FMCADâĂŹ06*, (August):68–75, 2006.

[12] C. Grelck, A. Shafarenko (eds):, F. Penczek, C. Grelck, H. Cai, J. Julku, P. Hölzenspies, S.B. Scholz, and A. Shafarenko. S-Net Language Report 2.0. Technical Report 499, University of Hertfordshire, School of Computer Science, Hatfield, England, United Kingdom, 2010.

[13] C. Grelck, S.B. Scholz, and A. Shafarenko. Asynchronous Stream Processing with S-Net. *International Journal of Parallel Programming*, 38(1):38–67, 2010.

[14] B. Heidergott, Geert Jan Olsder, and Jacob van der Woude. *Max Plus at Work: modeling and analysis of synchronized systems*. Princeton University Press, 2006.

[15] Philip Kaj Ferdinand Hölzenspies. *On run-time exploitation of concurrency*. PhD thesis, Enschede, April 2010.

[16] E.A. Lee. *Interaction of finite state machines and concurrency models*. IEEE.

[17] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245.

[18] Silvano Martello and Paolo Toth. Knapsack problems: algorithms and computer implementations. November 1990.

[19] Orlando Moreira, Jacob Jan-David Mol, and Marco Bekooij. Online resource management in a multiprocessor with a network-on-chip. *Symposium on Applied Computing*, 2007.

[20] M. Nakamura and M. Silva. *Cycle time computation in deterministically timed weighted marked graphs*. IEEE.

[21] Stuart J. Russell and Peter Norvig. Artificial intelligence: a modern approach. January 1995.

[22] Sundararajan Sriram and Shuvra S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization, 1st edition*. First edition, 2000.

[23] T D ter Braak. Run-time Spatial Resource Management in Heterogeneous MPSoCs. Master's thesis, Univ. of Twente, August 2009.

[24] B.D. Theelen, M.C.W. Geilen, T. Basten, J.P.M. Voeten, S.V. Gheorghita, and S. Stuijk. *A scenario-aware data flow model for combined long-run average and worst-case performance analysis*. IEEE.

[25] Maarten H. Wiggers, Marco J.G. Bekooij, and Gerard J.M. Smit. *Buffer Capacity Computation for Throughput Constrained Streaming Applications with Data-Dependent Inter-Task Communication*. IEEE, April 2008.