ARCHITECTURAL PROJECT: Architecture Exploration (SPADE 2)

REPORT: 2

SUBJECT: Research Problems in System-level Performance Analysis

DATE: October 23rd, 2001

ABSTRACT: In this document we report on research challenges of system-level exploration. We see two possible approaches to deal with these exploration challenges: the trace driven approach and the control data flow graph approach. We briefly describe the most general characteristics of both of them. The research challenges are introduced and positioned in a system-level exploration flow.

AUTHOR: Vladimir D. Živković, LIACS - Leiden University

KEYWORDS: Design-space exploration, internal application model, internal system model, trace-driven, control data-flow graph
1 Introduction

It is a very usual requirement today that the complex signal processing applications are to be designed and implemented as embedded computing systems. Thus, these systems must meet many constraints (e.g., real-time constraints), and have to provide sophisticated functionalities (e.g., to perform complex algorithms over a large amount of data). In addition to these, they have to be featured with low manufacturing costs, as well as with low power consumption. Finally, most embedded computing systems are designed by relatively small teams on tight deadlines. Building a cycle-accurate model of a complete system including the optimisation and compilation of software components is a huge effort. Moreover, when building such a model a lot of details get fixed, leaving less room for optimisations. Therefore, system-level methodologies and tools that allow early exploration of a wider design-space, and, give cost effective designs, are preferred. In this document we report on research challenges of system-level exploration.

This report is organised as follows. First, we give some general remarks about the scope of our research problem in Section 2.1. Then we briefly describe two possible exploration approaches in Section 2.2. After that, we define the main roadmap entities in Section 2.3. Then we describe each of the challenges in Sections 2.4 through 2.8. Finally, we draw some conclusions in Section 3.

2 Problem Statement

2.1 Defining the Scope

In Archer we are interested in signal-processing systems. We want to explore these systems and related architectures at the higher design-level, called system-level. We are focused on stream processing applications. Such applications can be modelled using the Kahn Process Network (KPN) Model of Computation (MoC). In this MoC application processes communicate via unbounded channels by performing blocking read operations and non-blocking write operations. The KPN MoC is deterministic, i.e., for the given input the same output is produced, irrespective of a schedule. YAPI is a tool that supports the modelling of applications as KPNs. In addition, YAPI offers a construct that enables the modelling of non-deterministic functional behaviour. The YAPI select operation explicitly models the influence of the scheduling on an application execution.

There are two major objectives that can be worked on here, and they are illustrated in Figure 1:

1. exploration of architectures and mappings driven by application models, where the aim is to obtain performance numbers starting from the YAPI C code of the application models (horizontal bidirectional arrows), or
2. system synthesis driven by application models, where the aim is to generate the C code of an implementation from the YAPI C code (vertical arrow that passes through different levels of abstraction).

Since Archer is an architecture exploration project, we are interested in the first objective.

2.2 Two Possible Exploration Approaches

During design space exploration we need support for mapping application models onto architecture models in order to evaluate the performance of different application-architecture combinations. Starting from YAPI application models, certain mapping steps need to be performed in order to support
explore with sufficient accuracy. For example, when mapping YAPI models onto a shared memory architecture, we need to separate the synchronisation and data transfer using finer grain primitives than \textit{read} and \textit{write}.

We see two possible approaches for implementing such mapping steps in an exploration environment. We call these approaches the \textit{trace driven} approach and the \textit{Control Data Flow Graph} (CDFG) approach. In both cases, there is a clear separation of an application and an architecture model, and they are related through a mapping layer.

In a \textit{trace driven} approach, an application model is executed on a single data set, after which we can abstract from the exact functional behaviour and we use abstract traces in the mapping steps and architecture simulation. Traces represent the unrolled application execution, where a single trace is devoted to the execution of a single application process. Each process is executed on top of a processing unit in the architecture. The architecture model is non-functional in that it does not process application data. See Figure 2, left side. In this approach the mapping takes as input an execution trace of the application and produces a refined trace that is used to stimulate the execution of a non-functional architecture model. During the execution of the architecture model, run-time scheduling may be performed.

In SPADE, the trace driven approach has been used because then it is very simple to simulate deterministic applications. Since traces are derived by the application execution on a single data set, the system model can be derived relatively easy. However, if the non-deterministic applications are to be simulated, the trace driven simulation is likely to become more complex (Section 2.4).

In a \textit{CDFG} approach, the application model is captured in a representation that preserves all application functionalities, including repetition and control constructs. These representations will take the form of Control Data Flow Graphs (CDFGs). Application functionalities must be fully preserved since the data set still needs to be processed in the architecture simulation. See Figure 2, right side. The CDFGs are derived by parsing and by transforming the source code, e.g. by compiling the source code. In this approach the mapping takes as input a CDFG and produces a refined CDFG that is used in the architecture simulation. During the execution of the architecture model, run-time scheduling may be performed. Manipulating CDFGs may be more complex than manipulating traces.

The two approaches can also be qualified as follows: in the trace driven approach the mapping step
manipulates traces, whereas in the CDFG approach the mapping step manipulates a trace generator. In both approaches the architecture models can have varying degrees of abstraction. The CDFG approach somewhat resembles a flow for system synthesis, as the refined CDFG may be used to generate refined application code that can be used upon implementation.

![Diagram showing two exploration approaches](image)

Figure 2: Illustration of two possible exploration approaches

While trace driven simulation has been our approach in the past, and some initial extensions have been explored in Archer, the CDFG approach has only recently considered as an option. We have not yet decided whether to proceed with the trace driven approach or to switch to the CDFG approach. Further examination is required to come to a choice. In any case, we will evaluate both options by going through a small example case in order to come to a good decision.

### 2.3 Defining the Main Roadmap Entities

In both approaches presented in Section 2.2, the exploration has been starting from an application model and, after an explicit mapping step, it has ended with the architecture execution (see Figure 2). It is exactly the mapping step that generates differences between a model seen before and after this step. For example, although the resource capacity is a crucial property of an architecture, no information from the application related to that property is needed. However, application and architecture features are by nature different, and, therefore, the information provided by the application model (YAPI) may not be expressed in a way the architecture deal with. Hence, apart from application models we also have to consider models that capture the mapping of the application onto the architecture. As a result, two important entities in the exploration flow can be established:

1. **An internal application model**, i.e., a data-structure that captures the information from the YAPI model execution, and
2. **An internal system model**, a representation that contains all the relevant information for the system execution.

Deriving the internal system model from the internal application model is driven both by the architecture specification and the explicit mapping choices. Depending on what specifications are given and how they are given, different system models can be acquired. The execution of system models simulates the execution of the real system. In the trace driven approach system model is only
valid for the data-set that was used upon the execution of the YAPI application model (Figure 2, left side). As a result, different performance numbers can be acquired, different scheduling techniques can be tested, etc. This information is crucial to direct the system synthesis, that we have mentioned in Section 2.1.

With Table 1 we want to establish some general notions that come from the terminology we have introduced so far. It shows what is the result of “the product” of the main exploration approaches and the main exploration entities. Therefore, when we speak about “traces” we mean a trace-driven approach, and when we speak about control data flow graphs we mean a CDFG approach.

<table>
<thead>
<tr>
<th></th>
<th>Trace Driven Approach</th>
<th>Control Data-flow Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Internal Application Model</strong></td>
<td>traces</td>
<td>CDFGs</td>
</tr>
<tr>
<td><strong>Internal System Model</strong></td>
<td>refined traces + bus sharing</td>
<td>refined CDFGs</td>
</tr>
</tbody>
</table>

Figure 3 illustrates the exploration trajectories and highlights challenges in the area of system-level modelling and architecture exploration. The next subsections clarify the challenges given in Figure 3.
Figure 3: The relation between exploration and implementation in the context of system-level modelling
2.4 Challenge 1: Supporting Non-deterministic Application Models

SPADE is a system-level methodology and tool that uses trace driven simulation for exploration purposes. SPADE supports exploration of architectures and application mappings for deterministic applications. The non-deterministic YAPI select function is not supported in SPADE. This is because the application model so far has been the Kahn Process Network model that is deterministic by definition. However, the realistic world is non-deterministic, and therefore, system models have to be capable to capture this behaviour. The question that arises here is: How can we capture the non-determinism that the YAPI select introduces?

The outcome of the select will depend on the time behaviour of the architecture. Therefore, we cannot just unroll processes, but we must evaluate the outcome of selects using information on the state of the architecture. Once the select outcome in the architecture is known, this outcome causes communication to the application, which can then proceed with its behavioural execution.

Conceptually, there are two options to deal with the YAPI-select:

1. “unroll” a process until a select trace event has been encountered, and then suspend the execution of that process in the application model (trace-driven approach), or

2. compile the application model with all the repetition statements and the control statements and derive the CDFGs that take YAPI-selects into account (CDFG approach).

![Diagram](image)

Figure 4: Co-simulation approach: block the trace unrolling of process C until architecture signals
In the first case, we only slightly extend the working of the application/architecture co-simulation we have advocated so far. Every YAPI-select simply stops the application process until the architecture signals the resuming condition. The relationship between an application model, an architecture model, and mapping is shown in Figure 4. The numbering scheme in Figure 4 shows the sequence of actions when a YAPI-select is encountered in the trace of process C.

In the second case, YAPI-selects are captured in the CDFG representation, where some control has to be evaluated in order to follow the right execution choice. Therefore, in the second case, application functionality is transformed into the CDFGs that comprise the evaluation of the control and repetition events from the application model. Since the internal application model in this case becomes CDFGs, a question is whether we should abandon the trace-driven approach at the beginning of the exploration trajectory, and instead opt for the CDFG approach. In order to give good answer of this question, further investigation is needed, in a sense how both cases behave according to some criteria (e.g., complexity of the trace-driven simulation and the CDFG approach when the YAPI select is allowed in the application model). The second option is illustrated in Figure 5.

Dealing with non-determinism is entitled as Challenge 1 in Figure 3. Before we continue with the next challenge, we have to mention that keeping traces in an explicit way leaves the co-simulation concept alive.

2.5 Challenge 2: Capturing the Possible Parallelism from the Application Model

Currently, in SPADE, traces are constructed as totally ordered lists of application events (reads, writes, and executes), i.e., YAPI processes are considered to be strictly sequential. This is, again, consistent with the KPN model definition. Therefore, this trace concept leads to a conservative (in fact pessimistic) understanding of possible orderings of process events: no information about potential parallelism is communicated to the architecture. However, in the architecture specification we usually allow processing units to empower simultaneous execution of independent activities (e.g., parallel
read operations). On the other hand, these features cannot be fully utilised when the trace-events they consume are totally ordered. As a consequence, the results of a system simulation become inaccurate due to the inability to model a realistic mapping onto architectures. Questions that arise here are:

1. What kind of a dependency analysis of the application model should be done to reveal more parallelism?

2. Should the control constructs be also included in the internal application model?

3. How can the inter-process parallelism be included in the internal application model?

4. Which internal application model captures more parallelism explicitly?

These questions put a different light on the current understanding of internal application model, since these models are now constructed in a way such that they capture only the minimal needed information for the system simulation, which may not be sufficient. The consequence and severe limitation of this is that such architecture mimics the MoC of the application. For example, in the case of the trace driven approach, with more expressive trace events, more information about potential parallelism can be captured, and, therefore, application modelling and architecture modelling become less coupled. In Figure 3 this is illustrated as the Challenge 2. In the attached Memo5, we have reported on preliminary work in this direction.

### 2.6 Challenge 3: Architecture and Mapping Specification

SPADE uses a building-block approach for architecture modelling. Architecture components are generic building blocks that have only timing behaviour; functional behaviour is captured in the application processes. An architecture specification file in SPADE contains:

1. a **structural description** of the architecture in terms of abstract architecture components (processors, busses, FIFOs), their terminals (input ports, output ports), and connections among these terminals, and

2. a **set of parameters** that are directly related to the timing behaviour of each architecture component.

In order to instantiate appropriate architectures from an architecture specification, SPADE requests a wide library with generic modules. The TSS based library is used for this purpose.

The mapping specification in SPADE consists of:

1. an **explicit mapping** step, where each application process is mapped on a particular architecture component, and

2. a **scheduler policy selection**, for each architecture component onto of which more than one application process has been mapped.

Currently, SPADE supports only a heuristic scheduling algorithm with Round-Robin scheduling policy. Also, mapping of an application model onto the architecture model is in 1-to-1 fashion in terms of ports and channels available in an application model.

While architecture and mapping specifications in SPADE have their benefits (e.g., specifying new architectures and different mappings is straightforward), they also suffer from a serious drawback. All behaviours, such as a symbolic instruction execution, communication protocols, scheduling policies, are predefined (default) and alternatives cannot be expressed easily. Simply, in order to add new behaviours (e.g., an ASIP that can pipeline independent reads, executes, and writes) one has to be totally familiar with:
1. the trace and the mapping mechanism that SPADE uses,
2. the behaviour of other generic (TSS) architecture modules, and
3. the TSS programming.

The above illustrates that the abstractions that SPADE uses for architecture and mapping specifications are strictly limited to what is provided already in the TSS library, and what is supported by the SPADE platform.

We would like to give to the designers more freedom here. This is illustrated as Challenge 3 in Figure 3. If the notions of the internal application model and the internal system model are kept fixed, and the adapters that allow communication between these two internal models are available, then a lot of different possibilities of how this communication will take place can be explored. This can be seen as an exposure of the timing behaviour specification to a designer. He would be allowed either to select the library modules or to specify behaviours, e.g., in terms of FSMs, for each architecture component and mapping item. Of course, he would have to follow predefined input/output signals, that come/go from/to an internal application/system model. All this is considered in more detail in the attached Memo5, where we go through a bus-buffered interface example. Thus, apart from trying to fit the desired architecture or mapping specifications in what is available in the library of modules, designers would be allowed to express their own creativity by having the opportunity to simulate different component choices.

2.7 Challenge 4: Transformation Steps

So far, we have introduced the main exploration stages. However, we have not introduced a flow (or a set of transformations) that relates the stages one to another. This flow in the exploration trajectory, particularly from an internal application model to an internal system model, is not trivial. We have to apply the information that is collected from architecture and mapping specifications into the internal application model. This mangling or mixture must be done in a way that it results in correctness by construction of both logical and timing behaviours. Therefore, a conversion in an “all at once” fashion is out of the question. We have to detect what kind of transformations have to be applied here and in what order. This is illustrated as Challenge 4 in Figure 3.

For example, it is necessary first to transform application primitives (atoms) into the system-like primitives (atoms). Second, we have to introduce resource constraints (capacity). These two steps are depending on the architecture specification, and may be influenced by the the mapping specification. The next step, i.e., the arbitration of shared resources, is driven by the mapping specification. If all connections in an architecture network are dedicated, then this step can be skipped. Finally, for the case of an interface based design appropriate transformations have to be taken. Also, this step may be driven by the user defined specification of the system behaviour (see Section 2.6).

What is now available in SPADE-traces is just a special case of the exploration flow presented in this report. Both application and architecture trace models, that somehow correspond to the internal application and system models proposed in this report, are equal (both are based on the KPN MoC). For a particular design-space this is satisfactory. However, for the challenges that we want to address in Archer, the SPADE approach needs to be revisited. More about this can again be found in the attached Memo5.
2.8 Challenge 5: System Model

The last challenge in Figure 3 is indicated as the System Model. A system model, or more precisely an internal system model, is a representation that captures all relevant properties of a system, both static (dependencies, synchronisation) and dynamic (arbitration, scheduling). It is the product of transformation steps previously applied on an internal application model. It comprises all aspects that these transformation steps have addressed (e.g., synchronisation, buffering, handshaking over synchronous communication channels, communication network arbitration, communication network protocols). By executing the internal system model on top of the communication network model synthesised from the architecture specification, the system simulation is accomplished. The challenge here is to precisely determine (identify) which aspects of the implementation need to be captured, and, also, how this has to be achieved. This is illustrated as Challenge 5 in Figure 3.

Generally speaking, different system models can be synthesised by different exploration trajectories in Figure 3. However, the CDFG oriented path is more implementation oriented and therefore, the system model there is valid for multiple data-sets. See Figure 2. On the other hand, we have addressed the trace-driven exploration path in Memo5.

3 Conclusion

In this document we defined our research problem. We have identified a number of challenges. We identified two possible approaches and are planning an example case study to select one of them.