An Evolutionary Approach to System-Level Synthesis

Jürgen Teich, Tobias Blickle, Lothar Thiele

Abstract—In this paper, we consider system-level synthesis as the problem of optimally mapping a task-level specification onto a heterogeneous hardware/software architecture. This problem requires (1) the selection of the architecture (allocation) including general purpose and dedicated processors, ASICs, buses and memories, (2) the mapping of the algorithm onto the selected architecture in space (binding) and time (scheduling) and (3) the design space exploration with the goal to find a set of implementations that satisfy a number of constraints on cost and performance. Here, a new graph-based mapping model is introduced to specify the task of system-synthesis as an optimization problem. An Evolutionary Algorithm is adapted to solve this problem and is applied to explore the design space of video-codex implementations.

I. INTRODUCTION

A. Optimization Methodology

The proposed optimization methodology treats the problem of optimizing the mapping of a large-grain dataflow graph-based specification onto a heterogeneous hardware/software architecture. This problem requires (1) the selection of the architecture (allocation) among a specified set of possible architectures, (2) the mapping of the algorithm onto a selected architecture in space (binding) and time (scheduling), and (3) the design space exploration with the goal to find the set of optimal implementations that satisfy a number of constraints on cost and performance.

Our approach provides a) a new system-level algorithm/architecture model for heterogeneous hardware/software systems, and b) applies Evolutionary Algorithms to system-level synthesis showing that they can perform steps (1)-(3) in a single optimization run. Evolutionary Algorithms turn out to be a good candidate for system-level synthesis, because they a) iteratively improve a population (set) of implementations, b) they do not require the quality (cost) function to be linear (e.g., area-time product). In case Pareto-ranking is used as the fitness function, the design spaces can be explored in a single optimization run. Finally, c) they are known to work "well" on problems with large and non-convex search spaces, and d) they require a reasonable amount of run-time due to their heuristic nature.

B. Existing Approaches to System-Level Synthesis

There exist already many different approaches to system-level synthesis. Some approaches focus on control-dominant systems, e.g., [9], [6], and [10]. Other approaches deal with dataflow-dominant designs like [1], and [8]. The term system synthesis, however, is weakly defined. In some approaches, the result of the synthesis procedure is the description of a dedicated control & data path in VLSI, e.g., [13], [12], [14] or a multi-chip dedicated VLSI architecture, e.g., [11]. In other cases, it is a mixed hardware/software architecture. In some of these approaches, the architecture is already fixed, see, e.g., [6], or [8] and the synthesis problem is the problem of partitioning functionality into hardware and software blocks, see, e.g., [3], [13], [12], [14]. Some approaches do not handle communication as part of the synthesis problem [4].

II. THE PROBLEM OF SYSTEM SYNTHESIS

A. Graph-Based Specification

Definition 1 (Specification Graph) A specification graph is a graph $G_S = (V_S, E_S)$ consisting of D dependence graphs $G_i(V_i, E_i)$ for $1 \leq i \leq D$ and a set of mapping edges $E_M$. In particular, $V_S = \bigcup_{i=1}^{D} V_i$, $E_S = \bigcup_{i=1}^{D} E_i \cup E_M$ and $E_M = \bigcup_{i=1}^{D} E_{MI}$, where $E_{MI} \subseteq V_i \times V_{i+1}$ for $1 \leq i < D$.

Consequently, the specification graph consists of several layers of dependence graphs and mapping edges which relate the nodes of two neighboring dependence graphs. These layers correspond to levels of abstractions, for example algorithm description (problem graph), architecture description (architecture graph) and system description (chip graph). The edges represent user-defined mapping constraints in the form of a relation: 'can be implemented by'.

Example 1 Fig. 2 shows an example of a specification graph derived from a given problem graph (Fig. 1(a)), an architecture graph (Fig. 1(c)) derived from Fig. 1(b) and a chip graph (Fig. 1e) derived from Fig. 1(d). The edges between the two subgraphs are the additional edges $E_{MI}$ and $E_{M2}$ that describe all possible mappings. For example, operation $v_1$ can be executed only on $v_{RISC}$. Operation $v_2$ can be executed on $v_{RISC}$ or $v_{HW}$.

The specification allows the RISC processor, the hardware modules HW1, HW2 and the communication modules BR1, BR2 to be implemented in CHIP1. The communication BR1 can either be handled by CHIP1 or by the off-chip bus OCB.
implementation can be seen as the task of assigning activity values to each node and each edge of the architecture graph and/or chip graph. An activated mapping edge represents the fact that the source node is implemented on the target node.

B. System Synthesis

**Definition 3 (Allocation)** An allocation \( \alpha \) of a specification graph is the subset of all activated nodes and edges of the dependence graphs, i.e., \( \alpha = \alpha_V \cup \alpha_E \) with \( \alpha_V = \{ v \in V_S | a(v) = 1 \} \) and \( \alpha_E = \bigcup_{e \in E_M} \{ e \in E_\alpha | a(e) = 1 \} \).

**Definition 4 (Binding)** A binding \( \beta \) is the subset of all activated mapping edges, i.e., \( \beta = \{ e \in E_M | a(e) = 1 \} \).

**Definition 5 (Feasible Binding)** Given a specification \( G_S \) and an allocation \( \alpha \), a feasible binding \( \beta \) is a binding that satisfies the following criteria: a) Each activated edge \( e \in \beta \) starts and ends at an activated node, i.e., \( \forall e = (v, \bar{v}) \in \beta \), b) for each activated node \( v \in \alpha_V \) with \( v \in V_i \), \( 1 \leq i < D \) exactly one outgoing edge \( e \in V_M \) is activated, i.e., \( \{ e \in \beta | e = (v, \bar{v}) \in V_i \} = 1 \), and c) for each activated edge \( e = (v_i, v_j) \in \alpha_E \) with \( e \in E_\alpha \), \( 1 \leq i < D \) either both operations are mapped onto the same node, i.e., \( \bar{v}_i = \bar{v}_j \) with \( (v_i, \bar{v}_i), (v_j, \bar{v}_j) \in \beta \) or there exists an activated edge \( \bar{e} = (\bar{v}_i, \bar{v}_j) \in \alpha_E \) with \( \bar{e} \in E_{\alpha+1} \) to handle the communication associated with edge \( e \), i.e., \( (\bar{v}_i, \bar{v}_j) \in \alpha_E \) with \( (v_i, \bar{v}_i), (v_j, \bar{v}_j) \in \beta \).

The calculation of a feasible binding and therefore the test of an allocation for feasibility is hard. This result will influence the coding of allocation and binding in the EA.

**Theorem 1** The determination of a feasible binding is \( NP \)-complete.

**Proof:** By polynomial reduction of \textsc{boolean satisfiability}, see [2].

Finally, it is necessary to define a schedule. Here, we use the execution time delay \( \text{delay}(v, \beta) \) of an operation associated to node \( v \) of a problem graph \( G_P \). Note that the execution time depends on a particular binding \( \beta \).

**Definition 6 (Schedule)** Given specification \( G_S \) containing a problem graph \( G_P = G_P \), a feasible binding \( \beta \), and a function \( \text{delay} \) which determines the execution time delay of a node \( v \in V_P \). A schedule is a function \( \tau : V_P \rightarrow \mathbb{Z}^+ \) that satisfies for all edges \( e = (v_i, v_j) \in E_P \): \( \tau(v_j) \geq \tau(v_i) + \text{delay}(v_i, \beta) \).

Now, given a specification graph \( G_S \), a (valid) implementation is a triple \( (\alpha, \beta, \tau) \) where \( \alpha \) is an allocation, \( \beta \) is a feasible binding, and \( \tau \) is a schedule.

**Example 2** Fig. 2 shows an implementation of the specification shown in the same figure. The nodes and edges not allocated are shown dashed, as well as the edges \( e \in E_M \) that are not activated. The allocation of nodes is \( \alpha_V = \{ \text{vRISC, vHWM1, vBR1, vCHIP1} \} \) and the binding is \( \beta = \{ (v_1, \text{vRISC}), (v_2, \text{vRISC}), (v_3, \text{vHWM1}), (v_4, \text{vHWM1}), (v_5, \text{vBR1}), (v_6, \text{vRISC}), (v_7, \text{vBR1}), (v_8, \text{vBR1}), (v_9, \text{vCHIP1}) \} \). This means that all architecture components are bound to CHIP1. A schedule is
\[ \tau(v_1) = 0, \tau(v_2) = 1, \tau(v_3) = 2, \tau(v_4) = 21, \tau(v_5) = 1, \tau(v_6) = 21, \tau(v_7) = 4. \]

C. The Tasks of System Synthesis

With the model introduced above the task of system synthesis can be formulated as an optimization problem.

Definition 7 (System Synthesis) The task of system synthesis is the following optimization problem:

\[\begin{align*}
\text{minimize } h(\alpha, \beta, \tau), \\
\text{subject to } & \\
& \alpha \text{ is a feasible allocation,} \\
& \beta \text{ is a feasible binding,} \\
& \tau \text{ is a schedule, and} \\
& g_i(\alpha, \beta, \tau) \geq 0, \ i \in \{1, \ldots, q\}.
\end{align*}\]

The constraints on \(\alpha, \beta\) and \(\tau\) define the set of valid implementations. Additionally, there are functions \(g_i, i = 1, \ldots, q\) that together with the objective function \(h\) describe the optimization goal.

III. Optimization Method and Design Space Exploration

In this section, the application of an Evolutionary Algorithm (EA) is described to solve the problem of system synthesis as specified by Definition 7. An EA works on populations of individuals \(J_i, i = 1, \ldots, N\) where \(N\) is called size of the population, and each individual codes an implementation of the problem graph including an architecture and a mapping of nodes in the problem graph in space (binding) to that architecture. The EA consists of an optimization loop that applies the principles of reproduction, crossover and mutation to the strings that code implementations. The purpose is to iteratively find better populations. Each individual in an actual population \(P_k\) is ranked by evaluation of a fitness function that gives a measure how good an implementation is in terms of cost and performance, etc. The EA terminates after a certain number \(k_{\text{max}}\) of generated populations and outputs those implementations with the best fitness values.

In our case, the EA is responsible for the determination of the allocation and of the binding. As there are good heuristics known for scheduling, it is not necessary to load the EA with this task. This is why the schedule for the fixed architecture obtained by the EA is computed by a standard heuristic scheduler. This division of work is depicted in Fig. 3.

A. Coding of Implementations

A specification graph \(G_S\) may consist of \(D\) subgraphs \(G_i, 1 \leq i \leq D\) corresponding to \(D - 1\) mapping tasks. In order to handle these mapping tasks the allocation and binding of the levels is done sequentially from level 1 to level \(D - 1\). In each level the following steps are executed: First, the allocation of nodes \(V_{i+1}\) is decoded from the individual, next the binding of the edges \(e \in E_{Mi}\) is performed. In Fig. 4, the encoding of an allocation in the chromosome of an individual is shown.
above. The scheduler performs latency minimization under resource constraints. The scheduler returns the latency of the schedule and the start times of all nodes. Then, the complete fitness function can be evaluated for each individual.

B. EA Parameters

For design space exploration, the selection method should maintain a high diversity in the population. Therefore, we use restricted tournament selection. The specific encoding of an individual makes special crossover and mutation schemes necessary. In particular, for the allocation α, uniform crossover is used that randomly swaps a bit between two parents with a probability of 50%. For different priority lists, order based crossover is applied. The probability of crossover is 50%. A mutation of an allocation α consists in simply swapping the allocation bit with a probability of 50%. Mutation is applied to 20% of the individuals of a population.

C. The Choice of the Fitness Function

According to Definition 7, the system-synthesis task was described as an optimization problem. As the particular optimization goal depends on the specific design problem, the objective function \( h \) has to be designed individually for each problem. But the restriction to a feasible allocation and binding needs to be considered in any case. The fitness function \( f \) to be minimized is

\[
f(J) = \begin{cases} \sum_{i=1}^{N} \left( x_a(J) p_a + x_b(J) p_b : x_a(J) = 0 \lor x_b(J) = 0 \right) \\
\quad \quad h'(J) : \text{else} 
\end{cases}
\]

The \( p \) values are the penalty terms, i.e., \( p_a \) is the penalty term for an infeasible allocation and \( p_b \) for an infeasible binding. The boolean variables \( x \) denote whether the corresponding constraint is violated or not, e.g., \( x_a(J) = 0 \) if the allocation is feasible and \( x_a(J) = 1 \) if the allocation is infeasible. The values for the penalty terms \( p_a \) and \( p_b \) should be chosen such that any infeasible allocation or binding has a worse fitness value than any feasible implementation. The modified objective function \( h'(J) \) has to reflect the additional constraints \( g \).

In system-synthesis, usually many different criteria have to be optimized like for example cost, data-throughput, power consumption, and maintainability. The concept of Pareto-optimality gives a measure of concurrently comparing implementations to such criteria.

Definition 8 A point \( J_i \) is dominated by point \( J_k \) if point \( J_k \) is better than point \( J_i \) on all criteria, denoted as \( J_i \succ J_k \). A point (implementation) is said to be Pareto-optimal if it is not dominated by any other point.

In [7], a Pareto-ranking scheme is proposed for multimodal optimization. We obtain

\[
h'(J) = \sum_{i=1}^{N} \begin{cases} 1 : J_i \prec J \\
0 : \text{else} 
\end{cases}
\]

All Pareto-points in the population have an (optimal) fitness of zero. Note that the fitness of an individual depends on the population.

Example 3 As an example for Pareto-optimization consider the case of two-dimensional optimization with the criteria cost \( c(\alpha, \beta) \) and period \( P(\tau) \). One obtains as fitness function:

\[
h'(J) = \sum_{i=1}^{N} \begin{cases} 1 : (c(J) > c(J_i) \land P(J) \geq P(J_i)) \\
1 : (c(J) \geq c(J_i) \land P(J) > P(J_i)) \\
0 : \text{else} 
\end{cases}
\]

Here, \( c(J) \) is used as an abbreviation of \( c(\alpha(J), \beta(J)) \) and the dependence of \( \alpha \) and \( \beta \) on the individual \( J \) is explicitly denoted by \( \alpha(J), \beta(J) \). Similar \( P(J) \) stands for \( P(\tau(J)) \).

IV. Case Study

We explain our methodology using the example of a video codec for image compression using the H.261 standard. The problem graph of the coder is shown in Fig. 5. The synthesis problem is restricted to single level of hierarchy, i.e., only the mapping of the problem graph \( \mathcal{G}_P \) to the architecture graph \( \mathcal{G}_A \) is examined. Motion estimation is done by the block matching operation (BM), the block subtraction is named DIFF (difference) and the block addition REC (recover). Additionally, the quantization is split up into threshold calculation (TH) and quantization (Q).

The graph and a similar graph for the decoder (not shown here) is mapped onto a target architecture shown in Fig. 6. The architecture consists of three shared bus with different data rates, two memory modules (a single and a dual ported memory), two programmable RISC processors, a signal processor (DSP), several predefined hardware modules (namely a block matching module (BMM), a module for performing DCT/IDCT operations (DCTM), an subtract/adder module (SAM) a Huffman coder (HC), and I/O devices (INM and OUTM).

Possible mappings and delays of all modules are shown in Table 1. Due to space requirements, the mappings of communication nodes to resources are not specified here.

The number of possible bindings are \( 1.9 \times 10^7 \). Hence, the intractability of enumerative or exact methods to explore the search space.

The cost of an implementation is simply reflected by the sum of the costs of the allocated hardware components. The cost of each module is given in brackets in Fig. 6.
Fig. 6. Architecture template of architectures to implement the problem graph of the video codec. The values in brackets give the basic cost $c_b$ of each resource.

Table 1. The mapping of the functional nodes to architectural nodes.

Now, we aim to perform a design space exploration. For this purpose, the Pareto-ranking function introduced in Example 3 is used. The fitness value directly gives the number of implementations that dominate an individual $J$. With a population size of $N = 100$, the Pareto-set found in a single optimization run after 200 generations is shown in Table 2 and depicted in Fig. 7. In general, there is no evidence that the Pareto-points obtained above are the true Pareto-points of the problem. Surprisingly, all points in Table 2 turn out to be true Pareto-points of the problem. Fig. 8 shows the architecture corresponding to the fastest design point found. The minimal period of the implementation is obviously determined by the execution time of the block matching module (BMM).

V. CONCLUSIONS

The optimization methodology presented here is part of the CodeSign framework currently in development. The framework uses object-oriented high-level Petri-nets as the kernel language for the specification of heterogeneous systems. The complete framework including also interfaces to common design languages and code generation tools will be described in a separate paper.

References:

Table 2. Implementations obtained as Pareto-points in a single run of the Evolutionary Algorithm.

<table>
<thead>
<tr>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
<th>$J_5$</th>
<th>$J_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period</td>
<td>22</td>
<td>42</td>
<td>54</td>
<td>78</td>
<td>114</td>
</tr>
<tr>
<td>Cost</td>
<td>330</td>
<td>340</td>
<td>330</td>
<td>280</td>
<td>230</td>
</tr>
</tbody>
</table>