Abstract

In current embedded system design practice, only few architectural solutions and mappings of the functionalities of a system on the architecture’s components are examined. This paper presents an optimization-based method and the associated tool developed to help designers take architectural decisions. The principle of this approach is to efficiently explore the design space and to dynamically provide the user with the capabilities to visualize the evolution of selected criteria. The first objective is solved by developing an enhanced version of the adaptive simulated annealing algorithm. Since the method is iterative, multiple solutions may be examined and the tool lets the user stop exploration at any time, tune parameters and select solutions.

Moreover we present an approach for systems whose functionalities are specified by means of multiple models of computation, in order to handle descriptions of digital signal applications at several levels of detail.

The tool has been applied to a motion detection application in order to determine architectural parameters.

1 Introduction

The steady increase of integrated circuit densities during the last three decades has been the driving force enabling ever more sophisticated consumer products with a wealth of functionalities to reach the market. These products are most often embedded systems that become increasingly more difficult to design. The complexity of both the hardware and the software in these systems makes them so hard to debug that delivering the products on time is a real challenge. The decisions about system architecture, the use of dedicated components (FPGAs) and the selection of processors to be used as programmable components impact design complexity and hence design time and design cost. This process of defining the system architecture and of mapping its functionalities onto its components determines for which functionalities software will have to be developed and for which ones hardware must be designed.

In order to withstand high levels of system complexity, automatic construction techniques have been developed over the years (compilers, logical synthesis, place&route...). At the system level, architecture definition and application mapping cannot be fully automated, i.e. without taking into account the designers’ experience. At this phase of design many criteria must be taken into account (apportionment of design effort, time-to-market, monetary cost, reuse...) and these can not be expressed in a formalized manner. As a result decisions are often taken without much help from existing mapping techniques (see section 1.1). The exploration of the design space is quite limited, basically consisting in lengthy simulations of a few architectural configurations.

In this paper we present an interactive software environment that lets the designer visualize and traverse the design solution space (Fig. 1). The first objective is to develop the tool that will guide a user to the best design decisions. The second one is to propose a methodology for using this HW/SW partitioning method.

The methodology relies on four steps: characterization, exploration, implementation, and fine estimation that either provide informations to the exploration tool or refine the resulting solutions, as shown in Fig. 2.

The proposed methodology is described in section 2. This is not the main subject of the paper but presenting it helps understand the design context and the steps followed during our experiments (see section 6). These steps are supported by different teams and tools in a global project 1, the present paper focuses on the exploration method we developed.

The exploration algorithm is described in section 3 in a problem independent manner. Section 4 explains how the mapping problem (application, architecture) has been modeled. Because current systems exhibit a great variety of functionalities, our model is structured to handle a specification expressed in terms of a multiplicity of models of computation (MoCs). The emphasis in this paper is on a restricted set of models for DSP applications; these are task precedence graphs, data-flow graphs (DFG) and synchronous data flow (SDF) graphs. Our object-oriented

---

1 Epicure Project in our case study [1]
environment facilitates the development of modules associated to each new MoC that we integrate in our tool. The way of exploring the solution space for multi-model specifications is presented in section 5. Experiments and results obtained with the 4OM tool are presented in section 6. Finally, perspectives are given in Section 7.

1.1 Related Work

Architecture selection consists in two main types of decision : allocation and mapping. Allocation is the definition of architecture in terms of number and class of processors and hardware acceleration units. Mapping is the partitioning of functional primitives on this selected resources.

Much research has been done on automatic partitioning. Among the first existing tools, COSYMA[8] and LYCOS[7] find optimal solutions when partitioning applications on single processor architecture.

In order to provide better performance, technology offers multiprocessor SoCs that allow to exploit more efficiently coarse grain parallelism. The SpecSyn tool proposed since 1993 [6] supports multiprocessor architectures but assumes manual allocation. Moreover, when considering the mapping selection process, SpecSyn proposes only one solution to the designer, and thus is not suited to our exploration objective. More recent approaches treat the design space exploration problem for SoCs with multiple processors and HW accelerators, and consider both allocation and mapping selection. These approaches use either exhaustive search [15], by iteratively adding resources when constraints are violated, or optimization heuristics. In this last case, the optimization criterion can be a function of chip area [11], or of the cost of the computing units [4]. These criteria give a very partial view of the allocation consequences as discussed below, and a good optimization criterion would be very difficult to estimate [2] [5].

Moreover most of the algorithms used in current exploration tools depend on parameters (in the cost function or in the algorithm itself) and need long tuning phases that make their practical use complicated.

1.2 Contributions

In [13] we described in details the results obtained with our exploration method on a motion detection application. In view of the existing codesign methodology and the corresponding partitioning tools, our objectives in this paper and our contributions in the actual technological and research context are to propose a

- designer based exploration method where the objective is to satisfy designer requirements. This is ensured by an iterative optimization method (simulated annealing), presented in section 3 and an interactive graphical interface that we will describe in section 6.

- flexible exploration environment. For the algorithmic part, the optimization method is based on an adaptive version of simulated annealing, and for the specification part, our method aims at considering heterogeneous applications. For this reason our input specification is assumed to be composed of multiple models of computation.

- global exploration methodology which provides the practical utilization context of our tool. This methodology was followed during our experiments.

2 Methodology

We follow a standard (part of) top-down design flow, including specification, design space exploration and hardware/software co-design steps. The paper focuses on the HW/SW partitioning phase, but details about specification (before partitioning) and performance estimation (after partitioning) give an overall understanding of the way we realized exploration with our tool in the experimental stages. The approach is built to exploit the relevant information at each abstraction level and then to minimize design effort.

2.1 Characterization

This phase which provides basic informations about performances, area, memory, power consumption is a critical point of most partitioning approaches since decisions taken during the exploration step depend on the provided informations. The practical case study used during experiments is part of the EPICURE project. In this context, the well known target architecture (ARM9 processor + Virtex IC) allowed the use of associated design software (compilers), to perform simulations and then to extract a data base containing information about various implementations of primitives. In this dedicated context, multiple implementations of a functionality where described on the HW unit with different couples (execution time, number of configurable logic blocks). Details about applications characterization can be found in [13]. The need of relevant esti-
 mates, and then of dedicated tools for architectural components is another motivation to limit the capacity of our exploration method to fixed allocation design problems.

2.2 Implementation model

The solutions obtained when solving the partitioning problem are abstract in that the same solution may be used with different implementation models. A classification of these models is proposed in [16]; the more the task lengths depend on data values, the more dynamic (vs. static) decisions must be made, and the more the control (HW or SW) is complex. Models are—following that order—fully-static (FS), ordered-transaction (OT), self-timed (ST), quasi-static, static-assignment and fully dynamic. The FS model applies when task durations are data independent or their largest values are close enough to their average values to be used with little loss in throughput. Then, task starting times are determined at compile time (by exploration) and control is simple. As the task length variances increase, the OT model becomes relevant; the ordering of the task executions and of the communications is determined at compile time but not the starting times, which are determined at run time through explicit synchronization mechanisms (send and receive). Next, for higher variances, the ST model, for which only the ordering of the task executions is determined at compile time, becomes the most suitable.

The partitioning algorithm determines task orderings as part of the optimization process. Average performance is computed using average performance values for the functional objects obtained during the functionality characterization step that precedes architecture definition (during exploration). Variance and extreme values of performance of the functional objects are used to determine candidate implementation models (choice made at the level of the outer loop of Fig. 2).

2.3 Performance estimation

In general the performances of complex systems depend on the times at which their inputs are received and on the values of these inputs. Often there will be requirements not only on average performance but also on the probability of occurrence of poor (close to worst case) performance. To check that a solution found for the partitioning problem will satisfy low probability of poor performance requirements a statistical simulation is used. This is a refined global performance estimation step whose results—success or failure and by how much—are fed back to the architecture definition (optimization) step. The “architecture definition” computations (optimization inner loop) and the “performance estimation” computations (simulations), see Fig. 2, are of similar complexity. The feedback “law” (adjustment of safety margins) requires particular attention: the number of iterations of the outer loop must be kept small to avoid spending more than a few hours of computations on a workstation. In the case the rigorous real-time context of our experiments, a single global iteration of the methodology has been used.

3 Optimization method

3.1 Local search and simulated annealing

In order to efficiently explore design solution space we use a local search method which proposes a complete solution to the optimization problem at each iteration. This first characteristic is important since the method proposes a set of solutions during its computation in contrast with greedy approaches (such as list scheduling based algorithms) which build a single solution by means of specifically defined construction rules. For this reason, in our context local search seems to be the good candidate to perform an “exploration”. A local search algorithm starts from an initial solution, proposed or random, and modifies it iteratively; these modifications are called “moves”. A solution is accepted only if its “cost” is found lower than previous values, the search process will end when no local move decreases the cost further and the final solution is a local optimum. In order to converge to a solution close to a global optimum, we use a simulated annealing algorithm. A solution that increases the cost is then accepted with a probability that depends on a parameter: the temperature, denoted \( T \) (its inverse being denoted \( s \)). The way the temperature is controlled is called an annealing schedule; it gives the algorithm its convergence properties. A classical annealing schedule keeps temperature constant during steps of the order of a hundred iterations and then lowers \( T \) according to a fixed multiplicative factor (\( \Delta < 1 \)):

\[
T^+ = \Delta \cdot T.
\]

With this schedule, it is hard to tune \( \Delta \) to attain a desired quality of solution for a given problem instance and the
This motivated us to use an adaptive annealing schedule (section 3.2). We have developed an improved version of control move generation to maximize cooling speed while is poor.

Move generation affects the correlation between consecutive cost values and the adaptive schedule specifies how to control move generation to maximize cooling speed while satisfying the quasi-equilibrium condition. This version of simulated annealing has been used in VLSI circuit place and route tools [17]. We have recently improved on the estimation procedure (for \( \mu \) and \( \sigma \)) and also refined the selection of the moves. These modifications have been validated on several types of problems, including graph partitioning and function minimization as explained in [12], but this is out of the scope of this paper.

3.2 Adaptive simulated annealing

To perform a search that reaches a solution at most a few percent away from the global optimum, we have pursued the work carried out by LAM, who presented in [9] both an adaptive cooling schedule and a scheme for move selection that speed up significantly the convergence of simulated annealing. Adaptive SA employs a cooling schedule whose general form is independent of the optimization problem at hand. The problem’s cost function is viewed as the energy of a dynamical system whose states are the problem’s solutions. The schedule is obtained by maximizing the rate at which the temperature can be decreased subject to the constraint that the system be maintained in quasi-equilibrium. The adaptive nature of the schedule comes from the fact that it is expressed in terms of statistical quantities (mean \( \mu(s) \), variance \( \sigma(s) \), correlation \( r(s) \)) of the system’s cost function:

\[
s_{k+1} = s_k + \lambda \frac{1 - r(s_k)}{\sigma(s_k)},
\]

where \( \lambda \) is the parameter whose value controls the tradeoff between quality of solution and computation time is poor.

This motivated us to use an adaptive annealing schedule (section 3.2). We have developed an improved version of Lam’s adaptive SA algorithm [9] that provides solutions of a desired quality—the tradeoff “quality vs. computation time” being controlled by a single parameter—with very good statistical stability (i.e. final cost has a small variance).

### 3.3 Software architecture

The partitioning software environment is decomposed into three independent modules: exploration, estimation and specification, see Fig. 3.

Fig. 3. Software architecture of the exploration tool.

The exploration module, the heart of the environment (at the bottom of Fig. 3), implements the exploration and optimization algorithms and includes a constraint management module. The current optimization algorithm is used to optimize system performance but the “cost” module can be easily adapted to different criteria. This algorithmic engine is completely independent of the problem it solves: cost function and constraints are just data. Constraints are independent of the cost function (in contrast e.g. with [18] where they are integrated in the cost function) and the algorithm is adaptive so that it does not need the user to set or tune parameters according to the problem. Constraints are relaxed at the beginning of the exploration and hardened adaptively until they become strict, for the last iterations.

The set of moves embodies decisions to modify architecture, assignment and scheduling. The moves are tailored to each MoC and supplied by the specification module. They are devised so that any potentially interesting solution may be reached. If not interrupted, the algorithm converges toward a solution close to a global optimum [12].

### 4 Problem modeling

#### 4.1 Scenarios

Our HW/SW partitioning methodology must be able to handle a diversity of scenarios:

- The system architecture is set because an existing SoC from the previous generation of the product is reused. Only the assignment and scheduling of the tasks on the components have to be determined and just software will have to be developed.

- Some components of the architecture are known in advance (such as a licensed IP3 core) and parts of the application are allocated on these components at the outset. Partitioning must be solved for the remaining parts of the application with some components imposed.

- The product is new and an architecture that will achieve the desired performances has to be determined. A variety of solutions must be proposed according to the constraints and scenarios (e.g. investment in a particular IP license) contemplated by the project leader.

Although our method has been applied to the three classes

---

3 Intellectual Property
of problem, a great difficulty is the definition of good optimization criteria when allocation is not fixed, as mentioned above. Consequently, in the rest of the paper we consider that allocation is fixed (first and second scenario). In our tool, the requirements of a given scenario are stored in configuration files and the exploration process may be controlled by means of an interactive interface as discussed in section 6.

4.2 Architecture modeling

Since complex functionalities may have their primitives distributed on several components, we consider multiprocessor architectures. The processing elements (PEs) constituting the architecture share a communication medium (bus) characterized by a setup time, a bandwidth and an access time. They also share a main memory connected to the bus. Our method is not restricted to a particular target architecture since it can explore the types and numbers of programmable and dedicated computing resources in the system. When considering fixed allocation, there is no hardware development cost, and the method only uses software and reconfigurable hardware units (see Fig. 4) for which reuse is possible. The numbers and types of PEs in the system are supposed to be a requirement. Each PE executes instruction and manipulates local data both stored in dedicated memories. The proposed approach can be applied to a partially reconfigurable architecture insofar as the FPGA reconfiguration time depends on the number of CLBs needed to perform the desired computations. If a reconfigurable circuit does not permit multi-context execution, i.e., initiation of several contexts in parallel, reconfiguration cannot be overlapped with computations on the FPGA. An objective is to decompose the computations effected on the reconfigurable circuit into different contexts, and determine their sequential execution order. Moreover, each context can carry out one or more computational tasks, according to the numbers of CLBs required for the tasks. Performance estimates are made at compile-time, in particular communication latencies are statically evaluated as ordered transactions. From these estimates, the exploration method proposes mapping solutions. The performance model used during exploration follows a deterministic paradigm and the resulting architectural solutions correspond to statically defined schedules which allow global optimizations, and performance guarantees. As explained in section 2.2, a dynamic scheduling scheme can be refined from the ideal static solution.

4.3 Target applications

Current levels of integration lead to defining products with many secondary functionalities, witness for example the cellular phones. These supplemental functionalities originating in distinct applications (e.g., organizers) contribute to the complexity of embedded systems, which, consequently, are specified using a mix of MoCs. Our focus is on computationally intensive applications, often involving signal/image processing [13], for which achieving a good partitioning is critical. A model of computation of choice in that context is the SDF (Synchronous Data Flow) model. Many functions may be modeled with SDFs, such as recursive filtering, adaptive filtering in time and/or frequency domains, music synthesis, motion video coding and W-CDMA reception.

4.4 Exploration

The problem to be solved is to find a spatio-temporal mapping of an application, described by a task graph, onto the architecture described in section 4.2. We do this by exploring the space of feasible solutions, where a solution is defined by

- an assignment of the tasks on the processors (software) and the reconfigurable circuits (hardware),
- an assignment of the hardware tasks to time segments (logical contexts to be executed),
- an execution schedule for the software tasks and the hardware contexts, and
- a communication ordering, imposed by the task execution ordering.

When several tasks are assigned to the same resource, their execution order on that resource depends on the resource type (processor, ASIC, reconfigurable IC). Indeed, at the (coarse) granularity level considered here, software task execution is sequential—the processor exhibiting actual parallelism only at a finer level. At the other extreme, the computations for several tasks could be performed with maximal parallelism on an ASIC dedicated to
these computations. A dynamically reconfigurable circuit provides an intermediate solution since it can execute contexts both sequentially and concurrently; this corresponds to implementing a partial order for task execution. Therefore, a solution (see Fig. 4(b)) consists of

- a total order on the system processor,
- a globally total, locally partial (GTLP) order on the dynamically reconfigurable logic circuit, and,
- if there were an ASIC in the system, a partial order on that circuit.

Starting from a random initial solution the spatio-temporal mapping of a task is changed at each iteration. The corresponding move definition is described in [13]. After each move, the performance of the new solution is evaluated by determining the longest path in the modified search graph (Fig. 4(c)). Exploiting the property that simulated annealing is a local search method, the longest path may in some cases be obtained incrementally by means of a Woodbury-type update formula [3].

5 Multi-model specification

In general, for any application domain, several MoCs may be employed. Considering the signal processing domain, the most basic models are: data flow graph (DFG), precedence graph and SDF graph [10]. The specification module of the environment encapsulates the characteristics of these models, accordingly it includes for each model a specific module which provides the application module with the appropriate additional move definitions. A data flow graph is constructed out of two kinds of objects, the computation objects (tasks, nodes), executed on processing elements (PEs), and the data transfer objects (edges), effected on communication resources (CEs). The SDF model (Fig. 4.3(a)) may be viewed as an extension of the DFG model. Thus, it inherits the move definitions used to explore the solution space for the DFG model. In addition, other moves may be applied, permitting the exploration of a richer solution space. These are flattening, unfolding and retiming moves.

During an iteration a node \( i \) (actor) is executed \( q_i \) times; flattening that node amounts to replacing it by \( n_i \) nodes—where \( n_i \) divides \( q_i \)—that can be affected to different resources. Complete flattening, which amounts to replacing each actor \( i \) by \( q_i \) nodes, yields a homogeneous SDF, where each node is executed once per iteration (Fig. 4.3(b)). Unfolding an SDF graph \( m \) times amounts to replacing the graph by one for which one iteration corresponds to \( m \) iterations of the initial graph; each node is thus repeated \( m \) times. Flattening and unfolding provide more flexibility for assignment and throughput optimization (Fig. 4.3(c)) at the expense of control (code) complexity, hence of memory size.

Retiming of any of these graphs enables buffers to be shifted across nodes, modifying the tightness of the precedences (for instance an inter-iteration dependence may become intra-iteration while the converse may happen for another dependence); this widens the space of possible orderings. Assuming overlap between iterations for highest
Table 1. Average execution time and reconfiguration time (initial + dynamic), and average number of contexts and of tasks as functions of FPGA size.

<table>
<thead>
<tr>
<th>FPGA size (CLBs)</th>
<th>Execution time (ms)</th>
<th>( T_{\text{Total}} )</th>
<th>Number of contexts</th>
<th>Number of tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>77.4</td>
<td>0.5 + 0.7</td>
<td>2.8</td>
<td>4.5</td>
</tr>
<tr>
<td>200</td>
<td>41.6</td>
<td>1.0 + 6.1</td>
<td>8.3</td>
<td>13.1</td>
</tr>
<tr>
<td>300</td>
<td>43.0</td>
<td>1.4 + 3.9</td>
<td>6</td>
<td>16.8</td>
</tr>
<tr>
<td>400</td>
<td>22.4</td>
<td>1.9 + 11.8</td>
<td>9.2</td>
<td>15.4</td>
</tr>
<tr>
<td>500</td>
<td>19.9</td>
<td>2.4 + 11.9</td>
<td>7.7</td>
<td>14.9</td>
</tr>
<tr>
<td>600</td>
<td>23.9</td>
<td>3.0 + 12.2</td>
<td>7</td>
<td>18.3</td>
</tr>
<tr>
<td>700</td>
<td>19.7</td>
<td>3.4 + 12.8</td>
<td>6.7</td>
<td>20.4</td>
</tr>
<tr>
<td>800</td>
<td>18.4</td>
<td>3.4 + 12.7</td>
<td>6.4</td>
<td>18.2</td>
</tr>
<tr>
<td>900</td>
<td>21.1</td>
<td>4.2 + 14.7</td>
<td>5.9</td>
<td>18.6</td>
</tr>
<tr>
<td>1000</td>
<td>20.0</td>
<td>4.2 + 14.4</td>
<td>5.9</td>
<td>16.0</td>
</tr>
<tr>
<td>1500</td>
<td>21.2</td>
<td>4.6 + 13.1</td>
<td>4.2</td>
<td>19.6</td>
</tr>
<tr>
<td>2000</td>
<td>25.8</td>
<td>10.0 + 12.2</td>
<td>3.5</td>
<td>19.1</td>
</tr>
<tr>
<td>3000</td>
<td>28.7</td>
<td>15.0 + 7.0</td>
<td>2.6</td>
<td>22.4</td>
</tr>
<tr>
<td>4000</td>
<td>29.2</td>
<td>19.1 + 4.3</td>
<td>2</td>
<td>20.9</td>
</tr>
<tr>
<td>5000</td>
<td>33.5</td>
<td>24.5 + 0</td>
<td>1</td>
<td>23.5</td>
</tr>
<tr>
<td>10000</td>
<td>35.9</td>
<td>27.3 + 0</td>
<td>1</td>
<td>24.2</td>
</tr>
</tbody>
</table>

performance, the throughput is evaluated by constructing the associated inter-processor communication (IPC) graph and calculating the maximum cycle mean, whose inverse is the estimated throughput [16] (substituting performance module in Fig.3).

6 Experiments

In order to assess the effectiveness of our method, we have applied it to the motion detection application as described in [13]. The application performs object labeling with a real-time constraint of 40 ms per image. Since a software implementation on an ARM922 processor leads to an execution time of 76.4 ms, some portions of the application must be hardware-accelerated to meet the time constraint. The target architecture consists of an ARM922 processor and a reconfigurable FPGA of the Xilinx Virtex-E family. Several estimates are provided for each task on the FPGA, thus allowing exploration of the trade-off between number of CLBs and execution time [1].

Our method explores solutions characterized by a spatial partitioning, a temporal partitioning and a sequential ordering of the FPGA contexts. Results of exploration of the solution space are presented in [13], where we examined the impact of the device size (number of CLBs) on the performance that can be achieved (see Fig. 6). As a byproduct of this study we determined the size of the smallest device for which the 40 ms constraint is attained. Every value tabulated, and displayed, is an average of the results obtained for 100 runs. For each of the device sizes considered, the average values of the execution time, the reconfiguration time and the number of contexts are shown. As the size increases, the execution time drops quickly once the number of CLBs is large enough for a context to hold more than one task to be executed, since parallelism is then provided within the hardware. A minimum execution time is reached for about 800 CLBs and, as size increases further, execution time grows slowly and reaches a plateau around 5000 CLBs. Indeed, from this size up, all the hardware tasks can be executed in a single context. For such large devices, the optimization method starts from random solutions with one context and an execution time exceeding 75 ms to finally reach solutions with a single context as well but whose execution time is more than halved. The computation time on a desktop computer was about 10 seconds for each device size considered. Our tool (Fig. 7) allows to visualize selected optimization criteria, to stop/continue exploration, to analyze the current explored solution, to select exploration speed, to understand the conditions for satisfying performance constraints for example by partially modifying solutions (by manually moving one or more tasks). These practically important features are hard to exhibit. The distribution of automatically found solutions is shown in Fig.6.

7 Conclusion and perspectives

Replacing designer experience with automated design and selection procedures at the system level is not always possible. At the partitioning level, criteria such as monetary cost, design effort, time-to-market are hard to evaluate and combine. This is a practical limitation for automatic allocation and mapping selection tools. The approach presented in this paper only attempts to present the design solution space to the final decision-maker and help him explore that space. To match this goal, the paper presented a flexible optimization method based on an adaptive version of simulated annealing, an heterogeneous application specification (and the way to map such a description onto architectures) and the methodology used during
experiments. The interactive 4OM tool allowed to help designer during specific recongurable architecture sizing in a complete design project.

We are currently [14] working on real-time specification models with multiple tasks and heterogeneous periods and deadlines. Refinement of the actual modeling (as memory size constraints) is another way for improving our method.

Figure 7. The GUI of the exploration tool.

References