Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 23496, 20 pages
doi:10.1155/2007/23496

Research Article

Communication-Oriented Design Space Exploration for Reconfigurable Architectures

Lilian Bossuet,[1] Guy Gogniat,[2] and Jean-Luc Philippe[2]

[1] *Laboratoire de l'Intégration du Matériau au Système, Université de Bordeaux 1, CNRS UMR5218, 33405 Talence Cedex, France*
[2] *Laboratory of Electronic and Real Time Systems (LESTER), University of South Brittany, CNRS FRE2734, 56321 Lorient, Cedex, France*

Received 27 June 2006; Revised 21 December 2006; Accepted 16 January 2007

Recommended by Juergen Teich

Many academic works in computer engineering focus on reconfigurable architectures and associated tools. Fine-grain architectures, field programmable gate arrays (FPGAs), are the most well-known structures of reconfigurable hardware. Dedicated tools (generic or specific) allow for the exploration of their design space to choose the best architecture characteristics and/or to explore the application characteristics. The aim is to increase the synergy between the application and the architecture in order to get the best performance. However, there is no generic tool to perform such an exploration for coarse-grain or heterogeneous-grain architectures, just a small number of very specific tools are able to explore a limited set of architectures. To address this major lack, in this paper we propose a new design space exploration approach adapted to fine- and coarse-grain granularities. Our approach combines algorithmic and architecture explorations. It relies on an automatic estimation tool which computes the communication hierarchical distribution and the architectural processing resources use rate for the architecture under exploration. Such an approach forwards the rapid definition of efficient reconfigurable architectures dedicated to one or several applications.

1. INTRODUCTION

1.1. *Context of design space exploration for reconfigurable architectures*

Future applications like pervasive computing will require increasingly more flexibility. This major evolution will lead to imagine new execution platforms where flexibility, and also performances (speed, power consumption, throughput, etc.) will have to be guaranteed. Reconfigurable architectures correspond to an efficient solution to tackle this issue [1] as they are flexible and powerful, and represent a very attractive solution between software platform and dedicated hardware.

Many laboratories work on reconfigurable architectures [2] and propose different reconfigurable solutions. According to [3], the reconfigurable domain is a real jungle and it becomes mandatory to help the designer in order to increase the synergy between the application and the architecture. Applications will be efficiently implemented onto reconfigurable architectures only if several points are solved.

(i) Dynamic reconfiguration for efficient run-time adaptability: this point needs new techniques and tools like static and/or dynamic application partitioning tools, reconfiguration time estimation tools, control unit development, and operating systems to manage the reconfiguration steps.

(ii) Codesign of reconfigurable system on-chip: reconfigurable architectures are increasingly considered as a system on-chip, so they contain soft and dedicated hardware. Such systems need specific software and hardware design methodologies like codesign. These methodologies perform application partitioning and generally rely on performance estimation techniques to evaluate software and hardware implementations before covalidation and cosimulation of the design.

(iii) Design space exploration (DSE) for reconfigurable architectures: this last point focuses on exploring both the application and the architecture spaces. The aim is to find the most appropriate architecture for a single application or an applications family. In this case the architecture characteristics have to be defined according to reconfiguration issues. Such hardware architectures are not application-specific (like ASIC) and they

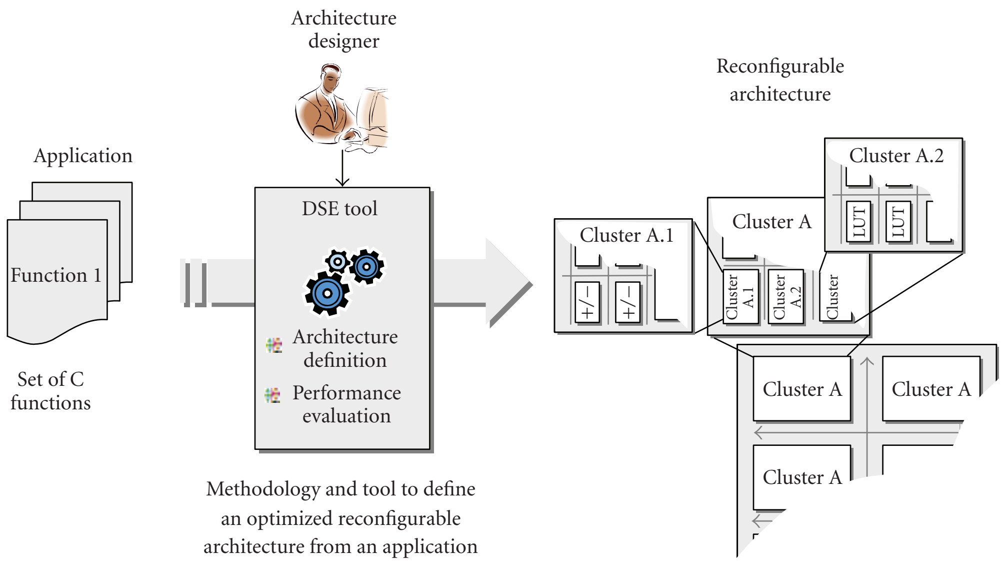


Figure 1: DSE process between application specification and architecture characterization.

provide large parallel structures that can be efficiently used within many applications. They embed coarse-grain and/or fine-grain operators and memories. New design techniques have to be developed in order to perform an efficient DSE for a better synergy between architectures and applications.

These points will have to be solved in the next few years in order to benefit from the huge potential provided by reconfigurable architectures. The challenges are developing operating systems to manage dynamic reconfiguration, developing codesign tools to build efficient reconfigurable SoC, and developing DSE tools to merge application space and architecture space. This paper focuses on the last challenge and demonstrates how to define an efficient architecture for an applications family.

1.2. Reconfigurable architecture DSE problematic

Although reconfigurable architectures correspond to hardware targets, their design is not the same as application-specific integrated circuit design (ASIC). Effectively, unlike ASICs that are designed to perform only one application with very tight performance constraints (area, latency, throughput, power consumption, etc.), reconfigurable architectures are designed to perform different applications relying on the same hardware capabilities. To be generic, reconfigurable architectures rely on massive parallel structures and use a dense reconfigurable routing network. They provide a set of low-level embedded elements (operator, logical function, memory block, etc.) organized into clusters. The DSE aim is to guide the designer to find some efficient clusters for an applications family. To address such an issue, it is essential to clearly define the architectural model under exploration.

(i) The template of the architecture for the exploration process is based on three hierarchical routing structures to propose the best communications scheduling inside the circuit and to take advantage of the application execution. Three levels of routing are generally admitted to represent an efficient solution. The low level of routing supports local communications between operators or logical elements and local variables storage, and the two high levels of routing support global communications.

(ii) The low level of the architecture relies on clusters, it is possible to use different clusters organized in a parallel structure. These clusters embed a range of coarse-grain arithmetic operators, coarse- and fine-grain logical operators, and memory blocks.

(iii) To be efficient, the architecture should take advantage of the application locality for treatment and storage. This last point is important to reach performance constraints.

As shown in Figure 1, DSE links the application specification (high-level specification) and the hierarchical clustered architecture. The DSE challenge is to take advantage of application execution graph to choose the best architecture parameters. Considering Figure 1, the exploration process will lead to the definition of the following:

(i) the type of resources within the low-level clusters (clusters A.2 and A.1),

(ii) the number of resources within the low-level clusters,

(iii) the number of low-level clusters within middle-level clusters (Cluster A),

(iv) the number of middle-level clusters within the whole reconfigurable architecture.

The exploration will thus enable designers to build their own architectures in order to be able to efficiently implement an application or an applications family. DSE gives designer information about application and architecture synergy, like inside-communications cost during application execution and the total use rate of the architecture to perform the application.

1.3. Contribution

The contribution of this work is to provide a new DSE method based on communications distribution inside the reconfigurable architecture in order to define a power-efficient architecture under a time constraint in synergy with an application (or an applications domain). This work permits the consideration of fine-grain, coarse-grain, and heterogeneous architectures for the same application. The designer can explore a large domain in the reconfigurable design space. An important characteristic of the exploration method is to consider a high level of description for both applications and architectures. In spite of estimation accuracy, it is possible to quickly find a hierarchical clustering for the architecture. Following the exploration process, the designer describes the application and the architecture with low-level specifications in order to use more specific tools and to design the final system.

1.4. Paper organization

This paper is organized as follows. Section 2 presents several works dealing with reconfigurable architecture DSE, and it gives a comparison table of these works. Section 3 presents the contribution and position of our work. Section 4 describes the application and architecture specifications used within the exploration process. In Section 5, the algorithms to estimate the communication distribution are proposed. Section 6 gives several results of exploration in the case of image computing and cryptography. Finally, Section 7 concludes this paper.

2. RELATED WORK

2.1. Introduction to design space exploration for reconfigurable architecture

It is possible to perform DSE at different levels of abstraction in order to progressively reduce the number of solutions. The more the abstraction level is refined, the more accurate the results are since a lower number of solutions need to be considered [4]. In the case of hardware DSE, two main methods are generally considered [5].

Synthesize and compare

This method uses a full synthesis flow to synthesize the application for each type of architecture under exploration before comparing the overall performance results. Using this method, it is possible to obtain very accurate performance measures. Nevertheless, it is necessary to have a specific synthesis tool for each type of architecture (which is not always available in the case of coarse-grain architecture exploration) or to use generic synthesis tools. However, synthesis steps compute very complex algorithms, which lead to a limited and slow exploration process. Furthermore, when using generic synthesis tools, it is necessary to have very good knowledge of the target architectures since it is necessary to develop a model for them. Hence, this method is not really adapted for a large and rapid architecture exploration and is more relevant for architecture refinement steps.

Estimate and compare

The second method relies on performance estimations instead of synthesis. In that case, it is necessary to consider a generic architecture model to describe the different target architectures. The goal is to perform relative performance estimations (speed, power consumption, and area) in order to compare different architectures very quickly. Although the estimations do not give necessarily real and accurate performance results, it is enough to compare the architectures since the relevant point in that case is that estimations are faithful and an absolute error is not the major concern.

These two methods are complementary and can be used within the same design process (same application) but at different abstraction levels. At a high level of abstraction, there are few synthesis tools and the architectural design space is huge. Therefore, it is more efficient to use an *estimate and compare* method in order to reduce the design space. At a low level of abstraction, the architectural design space is reduced. In this case, the exploration must converge towards a reliable architectural solution. Therefore, the *synthesize and compare* method is more relevant for this case. Obviously, a design space exploration flow should use several methods according to the level of abstraction. Interested readers can find more information about DSE in [6]. The following paragraph gives some examples of DSE tools and methods for FPGAs and coarse-grain reconfigurable architectures.

2.2. FPGA place and route generic tools used for DSE

Generic place and route tools for FPGAs are generally used within the *synthesize and compare* method. When the architecture model allows for the physical description of the routing structure, the tools can provide accurate performance estimates (particularly for speed and area). Such techniques provide interesting results concerning the use of the routing resources and the ability to route the device.

The versatile place and route (VPR) tool, developed at the University of Toronto in Canada, is a very interesting approach that works on a physical model (P-Spice model) [7]. VPR is a place and route tool that works at the logic level and is oriented for island style fine-grain architecture (like Xilinx FPGA). The physical model forwards the description of the architecture physical parameters (technology, routing type and size, routing switch resources, clusters size, etc.). VPR has an automatic mode which tries to route a circuit for different numbers of routing wires. Using VPR, it is possible to explore several aspects of the architecture like LUT and cluster size [8] or embedded memory size [9].

Madeot-Bet, a generic place and route tool developed by the University of Brest in France, uses a functional description of the architecture [10]. As VPR, Madeot-Bet is oriented for fine-grain reconfigurable architectures even if some extensions are currently under development to address coarse-grain architectures. The functional specification used to model the reconfigurable architecture enables the description of a large panel of architectures and is technological-independent. In fact, each element of the architecture is described by the functions it can execute. Although VPR and Madeot-Bet are generic place and route tools, they can be used for fine-grain architectural exploration, particularly for routing exploration.

2.3. FPGA exploration and estimation tools

According to the *estimate and compare* method, it is possible to perform DSE using estimation tools. The estimations can focus on one or several parameters (power consumption, speed, area, etc.) depending on the designer's expectations.

People of the University of Southern California in the USA present in [11] a power consumption estimation tool based on a parametric view. It provides a domain-specific modeling technique that exploits the knowledge of both the algorithm and the target architecture family for a given problem to develop a high-level model. This model captures architecture and algorithm features, parameters affecting the power performance, and several power estimation functions based on these parameters. However, the designer needs to have a great deal of knowledge in the domain (application and architecture) to be able to determine the parameters and the functions.

Enzler et al. [12] propose a high-level estimation methodology for area and speed developed within the Swiss Federal Institute of Technology. This methodology relies on the inputs and outputs, the control signals, the operators, the registers, the degree of parallelism, and the number of iterations within the application to characterize the application. With these characteristics, several parameters are computed to provide the delay and area performances. The target FPGA is characterized through the mapping of the operations. Therefore, a mapping model is specified for each type of operation from which the area and timing parameters are derived. The application and the target architecture are used to estimate the delay and area performances. After this estimation, several application parameters can be explored like the number of registers into the data path, the number of replications of a given block (parallelism), and the number of block decomposition into a sequence of identical subtasks (pipeline). This method mainly allows for the application exploration since the architecture exploration is rather limited.

2.4. Coarse-grain reconfigurable architectures DSE tools

Previous efforts focus on fine-grain reconfigurable architectures (FPGAs) even if it is possible to extend several techniques for coarse-grain architectures. Other works focus directly on coarse-grain architectures.

MIT researchers have developed a DSE framework for the raw microprocessor [13]. This reconfigurable architecture is reminiscent of coarse-grain FPGA and it comprises a replicated set of tiles coupled by a set of compiler orchestrated pipelined switches. Each tile contains an RISC processing core and SRAM memory for instructions and data. Several parameters, like the number of tiles, the memory size, or the communication bandwidth, characterize the architecture. The application is split into several subproblems, each of them is characterized by the number of architecture resources it consumes. Finally, some cost functions are used to estimate the performance (delay, area). This method is architecture-dedicated since the cost functions are only defined for one architecture. The result accuracy depends on the relevance of the architecture parameters.

In [14], the DSE flow targets a mesh architecture called KressArray [15], a fast reconfigurable ALU. The exploration tool Xplorer works at the algorithmic level and aims at assisting the designer to find a suitable architecture for a given set of applications. This tool is architecture-dependent, but the use of fuzzy logic to analyze the results of the exploration is a very attractive approach.

2.5. Comparative study

Table 1 gives a comparison between the related work presented above. The last line details our work characteristics in order to give the reader a comparison with previous efforts. The first two studies focus on simple FPGA (island style FPGA without embedded features like memory blocks or multipliers). Then, VPR and Madeot-Bet are really close since they use the same place and route algorithm and they mainly focus on the routing aspect. They can be used for exploration but it is necessary to first perform a high-level exploration to reduce the design space. The last studies are architecture-dependent and they are developed for coarse-grain reconfigurable architectures. Except for the second study, all of the studies explore the architecture according to one or several objectives. The second study explores the application algorithm and implementation. The two last studies have some automatic exploration steps since they are more specialized for a specific architecture. We can see on the last row that the different tools provide different results, depending on the starting design space (set of architectures or given architecture with a set of parameters). Therefore, they provide the best architecture in a set or the best configuration for a given architecture. Most of them give design information to help the designer to improve the application and the architecture definition.

VPR and MADEO can explore the largest design space since they are the most generic tools. The first two studies in Table 1 are specialized for an FPGA family, so the design space is limited. The last two studies are architecture-specific, so they can only explore a small design space. However, they provide very accurate estimations thanks to more accurate models. Therefore, the development of a DSE tool is a tradeoff between the design space size and the estimation accuracy.

Table 1: Characteristic comparison of the related work.

Tool	Architecture target	Applications specification	Architectures specification	Exploration	Results
Univ. Southern California [11]	Simple FPGA	Parameterization	Parameterization	Architectural objective: power manual and exhaustive	One architecture in a set (domain)
ETH [12]	Simple FPGA	Data flow graph	Characterization of simple operator	Algorithmic objectives: delay and area manual and exhaustive	Application design information (parallelism degree and level of pipeline)
VPR Univ. Toronto [7–9]	Complex FPGA	Netlist BLIF (or EDIF)	Structural model	Architectural objective: routing manual and exhaustive	FPGA architecture design information (clusters size, configurable element size, routing)
MADEO Univ. Brest [10]	Complex FPGA (multigrains)	Data flow graph	Structural model	Architectural objectives: routing, area and critical data-path manual and exhaustive	FPGA architecture design information (clusters size, configurable element size, routing)
Raw MIT [13]	Coarse-grain raw architecture	Parameterization	Parameterization	Architectural objective: execution time automatic and heuristic	Optimal architecture raw configuration
Xplorer Univ. Kaiserslautern [14, 15]	Coarse-grain KressArray architecture	ALE-X	Parameterization and structural model	Architectural objectives: power and routing automatic and heuristic	Optimal architecture KressArray configuration
Authors Univ. Bretagne Sud [16]	Heterogeneous architecture	HCDFG	Hierarchical functional model	Algorithmic and architectural objective: power automatic and heuristic	Architectural design information (cluster size, configurable element type, memory size, communication distribution, resource use rate)

This comparison helps us to define the characteristics of a new DSE method for reconfigurable architecture. It appears that all these methods are too specialized, technological-dependent, or architecture-dependent to explore a large design space with targets like fine-grain, coarse-grain or heterogeneous architectures. It is important to overcome this limitation to be able to compare fine-grain and coarse-grain architectures and to combine both in a single device (heterogeneous architecture). Furthermore, previous efforts require profound knowledge of the reconfigurable architecture and technology which can be difficult to handle for the designer. It would be helpful to provide a methodology where the designer can have an early estimation of his architecture performance without going through all the details of the architecture. All the above-presented tools do not take into account (static or dynamic) architecture reconfiguration during the performance estimation process. This aspect is becoming increasingly relevant and should be addressed within the exploration process.

Finally, the presentation of previous efforts relies on general benchmarks (which can also be used for ASIC implementation) to demonstrate their design flow and to validate the different concepts. In this paper, we use the same approach to validate our work as will be presented in Section 6.

3. A NEW VISION OF DSE BASED ON AN AUTOMATIC ESTIMATION TOOL FOR SOC DESIGN CALLED *DESIGN TROTTER*

3.1. The Design Trotter tool

The work presented in this paper is part of the Design Trotter project [17]. The Design Trotter framework is a computer-aided design (CAD) environment for reconfigurable system on a chip (RSoC). This environment is composed of several tools that work at different levels of abstraction and explore the design space in different ways. Figure 2 presents the interaction between the main tools of the Design Trotter framework. First, the application is specified using a subset of the C language, then the specification is translated into a hierarchical control data flow graph (HCDFG) [18]. For the present work, two tools of the *Design Trotter* framework are considered.

System estimation [19]

This tool aims at scheduling the application for several time constraints. The results are defined through "cost profiles," that is, scheduling for all the resources used by the

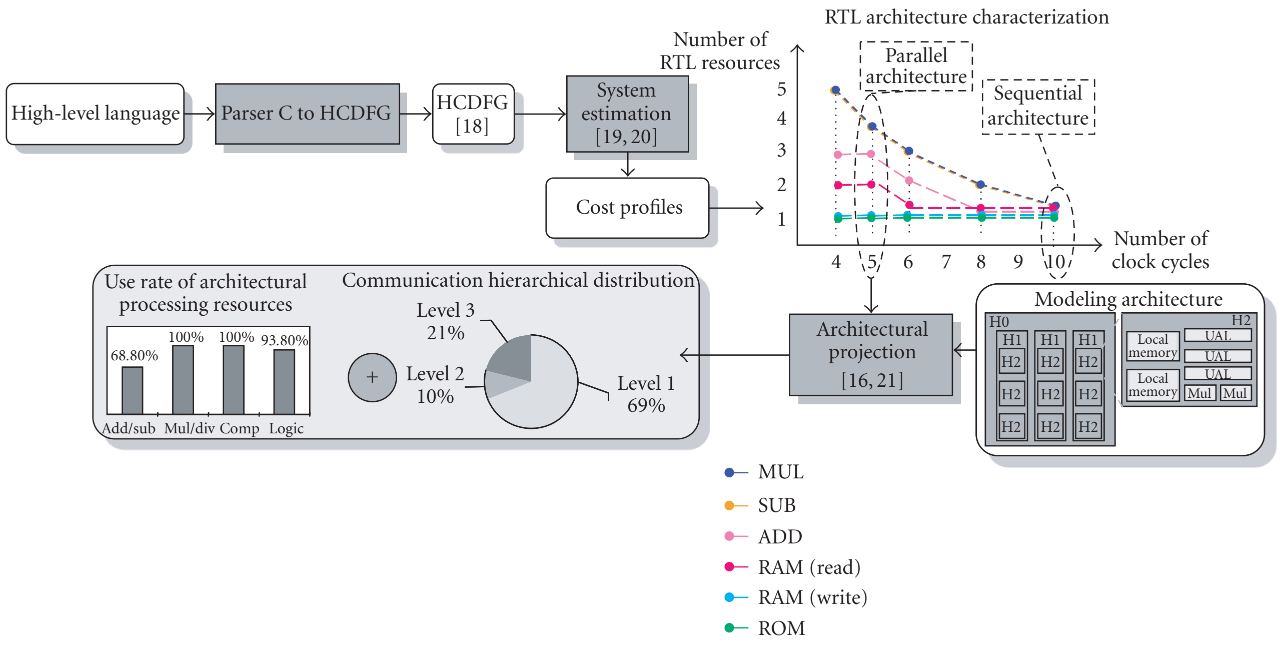


Figure 2: Design Trotter framework.

application. There is one cost profile for each time constraint. The processing and memory resources considered to compute a scheduling enable the definition of a logical architecture at the RTL level [20].

Architectural projection [16, 21]

This tool is used by the architectural exploration method presented in this paper. It computes performance estimations and enables the comparison between several architectures characterized by their power efficiency to implement an application. It provides design information that helps the designer to progressively improve the architecture definition through several iterations.

The system estimation tool performs an algorithmic exploration and the architectural projection tool drives the physical architecture exploration of the reconfigurable targets. So, the synergy between the application and the architecture is explored to reach the best couple application/architecture.

As shown in Figure 2, the cost profiles are the results of the system estimation tool and correspond to the inputs of the architectural projection tool. To launch the architectural projection tool, it is first necessary to select an RTL logical architecture to implement the application. The designer can consider a sequential RTL logical architecture with a high time constraint and a low number of resources (computing and memory resources) or the designer can consider a parallel RTL logical architecture with a low time constraint, so the number of resources is larger than for the sequential architecture. This last solution is adapted for hardware implementation (FPGA, coarse-grain reconfigurable architecture, or ASIC) as these technologies provide massive parallelism.

The cost profile of the selected RTL logical architecture provides the number of computing and memory resources for a given time constraint. To perform the system estimation, several scheduling algorithms have been developed in order to explore various tradeoffs depending on the characteristics of the application. The system estimation method (application metrics and scheduling techniques) is presented in [20].

The architectural projection tool provides the designer with use rate estimates of the architecture computing resources and the communication distribution for the different hierarchical levels of the architecture. For that purpose, the designer describes the target architecture with a hierarchical functional model. He can also refine the architecture description during the exploration process in order to tune the architecture parameters depending on the architectural projection results. The designer aims at finding a power-efficient architecture for his application under a time constraint.

In the next section, we detail the definition of *efficiency* since this notion drives the exploration process in our case. We also present the hierarchical functional model and the algorithms used within the architectural projection tool. The methodology developed to explore the design space is also presented.

3.2. Strategy of coarse-grain DSE

In order to develop a DSE methodology, it is necessary to emphasize some criteria to compare the architectures for the same application (and so the same RTL logical architecture). Our approach is under a time constraint since we consider an RTL logical architecture for a given number of cycles to perform the application as shown in the upper right part of Figure 2. According to [22], power consumption is a major

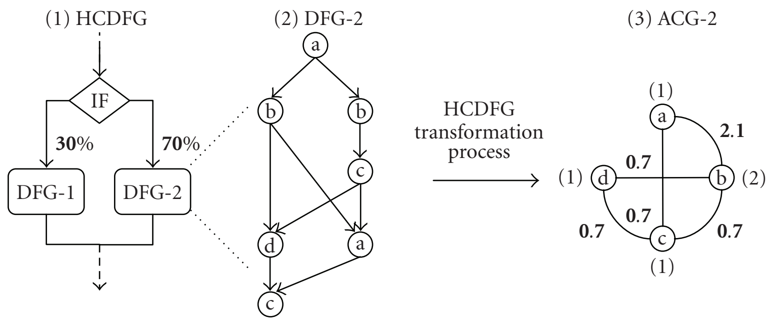


Figure 3: HCDFG transformation into ACG.

metric to compare the architecture efficiency under a time constraint. Power consumption reduction often becomes increasingly the main objective of a design flow, particularly for embedded systems. Power consumption is linked to the hardware time-life, the battery size, and the heat misbehavior.

In order to compare the power efficiency of different reconfigurable architectures, it is necessary to study the impact of the architectural resources on power consumption. We have studied this impact for fine-grain architecture (FPGA). According to our studies [23, 24] and according to other academic studies [25–29], several conclusions about power consumption of architectural resources for fine-grain architecture can be drawn.

(i) Routing resources are always the most consuming resources taking up from 50% to 80% of the total power consumption. However, the exact rate depends on design size, frequency, toggle rate of logical inputs, number of inputs/outputs, utilization rate of architecture resources, and synthesis option.
(ii) It is more power-efficient to use dedicated memory blocks instead of using distributed memories (e.g., with lookup tables).
(iii) A high use-rate for the architectural resources is better for power consumption since the free-resource static power leaks are lower.
(iv) It is very important to use local routing resources for intensive communicating resources (computing or memory). So the most communicating resources (which depend on the application specification) must be placed in a near neighborhood in order to decrease the routing power impact.

According to all these studies, computer-aided design tools must rely on a strategy of clustering for the most communicating resources within the architecture (for a given application or an applications family). So for a given application, a reconfigurable architecture has to be defined in order to promote an efficient clustering of the application resources. We extrapolate the previous conclusions of the fine-grain studies for coarse-grain and heterogeneous architectures since the internal routing structures are very similar, and the routing issues are still the same [27, 29].

The application and the architecture specifications have to emphasize the communications between the resources required by the application (according to the system estimation results) and the locality (from the routing point of view) of the architecture computing and memory resources. The next section presents these specifications and shows how they allow for the taking into account of communications and locality of the application and architecture resources.

4. APPLICATION AND ARCHITECTURE SPECIFICATIONS

4.1. Application specification

The application is first described using a subset of the C language [18]. This specification is then translated into a hierarchical control data flow graph (HCDFG) as an internal representation. This graph corresponds to a precise description of the application (computing, memory, and control) [18]. The system estimation tool provides information about the RTL logical architecture like the number of computing resources and memory resources needed for the application. As presented in the previous section, it is essential to obtain information about the communications between the application's resources since the most communicating resources have to be placed close within the architecture. To show this information, a new graph called average communication graph (ACG) has been developed. This particular graph highlights how each type of processing and memory resource communicates with each other. The edges in it represent the communications between two nodes and each node represents a type of processing resource or a memory resource (Figure 3).

Several differences exist between the HCDFG and the ACG graphs. The HCDFG describes the real control and data flow of the application independently of any implementation while the ACG corresponds to an approximation of the communications between operators and memories. The HCDFG is transformed into the ACG after having performed the scheduling of the operators and memories for a given time constraint. This scheduling as previously mentioned is performed during the system estimation step [20]. There are fewer nodes in the ACG than in the HCDFG since the ACG graph has only one node for one type of processing resources.

The ACG edges are not oriented and the communications are taken into account in all directions. Several attributes are added in the ACG to describe the internode communications.

Figure 3 shows an example of an HCDFG graph transformation into an ACG. In this example, the type of processing resource corresponds to a letter (a, b, c, or d). The number in brackets beside a node corresponds to the number of operators required for a given time constraint (result of the system estimation tool). The boldface number beside an edge is the total number of communications between two processing nodes. In order to define which pair of nodes communicates the most in the ACG, the relative number of communications between two processing types is computed. This value is obtained using the following equation:

$$\text{RelativeComm}_{\text{Op1}-\text{Op2}} = N(\text{Loop}) \times P(\text{Branch}) \times \frac{\text{TotalComm}_{\text{Op1}-\text{Op2}}}{\text{NumberOp1} + \text{NumberOp2}}, \quad (1)$$

where $\text{RelativeComm}_{\text{Op1}-\text{Op2}}$ is the relative number of communications, $\text{TotalComm}_{\text{Op1}-\text{Op2}}$ is the total number of communications, NumberOp1 and NumberOp2 are the numbers of allocated operators of each type. If the DFG is part of a hierarchical node with control nodes, $N(\text{Loop})$ is the loop number for a loop control node and $P(\text{Branch})$ is the branch probability for a conditional node as in Figure 3. For each control node, $N(\text{Loop})$ and $P(\text{Branch})$ are precomputed through a code profiling. For example, the ACG on the right-hand side in Figure 3 has two nodes, a and b, linked by one edge with a value equal to three. The node a describes one operator of a type and the node b describes two operators of b type. Therefore, the relative number on the edge, between these two nodes, is given by

$$\text{RelativeComm}_{a-b} = 0.7 \times \frac{3}{1+2} = 0.7. \quad (2)$$

4.2. *Reconfigurable architectures specification*

The reconfigurable architecture model is an important part of this contribution since it is a complex task to manage accuracy and high level of abstraction. According to Sections 1.2 and 3.2, the main characteristics of a model can be listed as follows.

(i) The model has to enable a large design space covering fine-grain, coarse-grain, and heterogeneous architectures.
(ii) The model has to describe the physical locality of computing and memory resources.
(iii) The model has to remain technologically-independent to be valid in spite of technological evolutions.
(iv) The model needs to be easily extended to take into account new architectural characteristics and possibilities.

To promote the architectural exploration, it is essential to mitigate the task of changing some architectural characteristics. The designer must be able to rapidly perform some manual evolutions of the architecture description.

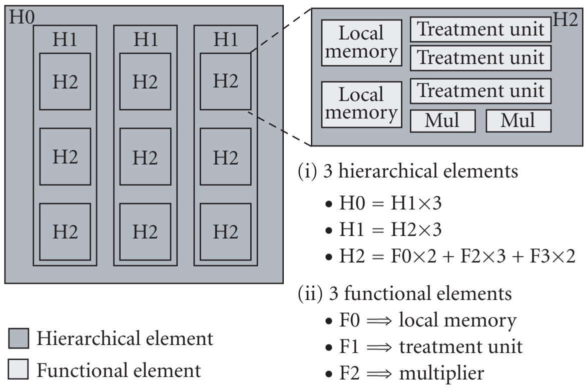


Figure 4: Example of coarse-grain reconfigurable architecture modeling with three hierarchical levels.

There are two possibilities to describe a reconfigurable architecture, using a physical description or a functional description. Using a physical description, as in [7], forwards the development of accurate estimations, but the model is technologically-dependent and cannot evolve easily. Moreover, the model can be complex (if the model describes all the details of the architecture) and it can be very tedious for the designer to manually change some architecture characteristics. Using the functional model, as in [10], leads to describe the architectural resources through the functions that they can realize (several functions if the resource is configurable). This type of model can easily evolve and the designer can quickly modify the architecture in order to explore the design space. Therefore, this kind of model is suitable for architectural specification.

According to Figure 1, in order to describe the architectural resources locality, we use a hierarchical view. For that, the proposed hierarchical functional model for reconfigurable architectures relies on two types of elements.

(i) *The hierarchical elements* are used to model the architectural hierarchy. They are containers; they embed other hierarchical elements or functional elements, and are described by their contents.
(ii) *The functional elements* describe the computing and memory resources. They are described by the list of functions that they can realize for a selected configuration.

Figure 4 shows an example of coarse-grain reconfigurable architecture modeling. In this figure it can be seen that according to the reconfigurable specification (see Section 1.2 and Figure 1), there are three hierarchical elements; H0 contains three H1, each H1 contains three H2. The high level of hierarchy is composed of one H0, the low level corresponds to the internal structure of H2. This last hierarchical element is composed of several functional elements. The architecture has three levels of hierarchy; the low level inside the H2 hierarchical elements contains only functional elements, the middle level inside H1 elements contains H2 elements, and the high level of the hierarchy is represented by H0.

This model uses two important hypotheses concerning the communication costs in the hierarchical elements. They enable the routing resources to be taken into account without using an accurate physical description. These hypotheses are as follows.

(i) The communication costs inside a hierarchical element are homogeneous. If the designer wants to describe large hierarchical elements, he must guarantee that this hypothesis will be verified with the use of dedicated routing resources in the corresponding hierarchical level of the architecture.

(ii) The second hypothesis is that the communications are less power consuming in the low level of hierarchy than in the high level of hierarchy. That is to say, for the architecture example in Figure 4, the communications inside the hierarchical element H2 (the communications between the embedded functional elements) consume less power than the communications inside the hierarchical element H1 (the communications between the elements H2). These latter communications consume less power than the communications inside the hierarchical element H0 and so between H1 hierarchical elements.

5. COMMUNICATION ESTIMATION AND EXPLORATION METHODOLOGY

5.1. *Introduction*

Our exploration method is based on an estimation of the communications hierarchical distribution within the architecture and an estimation of the architectural resources use rate. The goal of the exploration is to define an architecture promoting the clustering of the most communicating resources. The exploration method is interactive and is based on the architectural projection tool.

The architectural projection tool proposes an algorithm to merge the application ACG nodes according to their communications and to allocate the application resources within the low level. Since our approach works at the algorithmic level, it does not target a specific synthesis tool and does not consider any accurate physical architecture model. Instead of giving designers a single communication cost value that may present a significant absolute error due to backend synthesis algorithms and architecture refinement steps, we compute two bounds and an intermediate value. This approach gives the designer the ability to define the architecture at the algorithmic level with the guarantee that the final performance will belong to the estimated performance interval (Figure 5).

It also provides designers with metrics on allocation algorithm impact. Figure 5 illustrates this point, the architecture C has a narrow performance interval, so allocation algorithms will have a small effect on the final performance and a low complexity algorithm can be considered. The architecture B has a large performance interval, so allocation heuristic will have a strong impact on final performance and it might be important to consider better allocation algorithms.

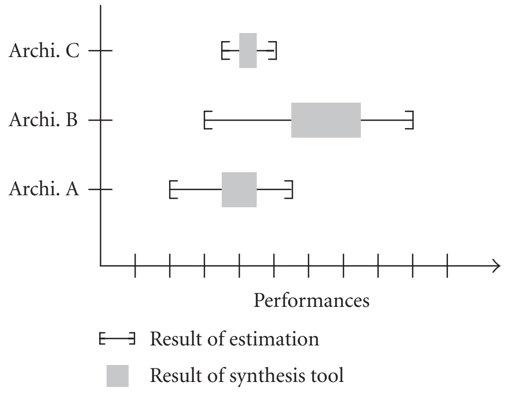


FIGURE 5: Bound performance results, example for three architectures.

To define such an approach, we have developed three algorithms that give three values of the communications hierarchical distribution estimation. The first algorithm, *the INTER algorithm*, gives distribution estimation with a maximum number of communications within the low hierarchical level H2, so the communication power cost for the application is minimal. The second algorithm, *the MIN algorithm*, gives distribution estimation with a minimum number of communications within the low hierarchical level H2. The last algorithm, *the INTER algorithm*, gives an intermediate value between the values given by the two other algorithms. Each algorithm is less complex and faster than an optimal algorithm.

The next sections present the different algorithms to perform the architectural projection and to obtain the communications hierarchical estimation. The tool deals with reconfigurable architectures composed of three levels of hierarchy since these levels are adequate in describing most current architectures.

5.2. *The architectural projection*

The architectural projection step makes the link between the required (application) and the available (architecture) resources with the challenge that the most communicating resources will be assigned in the same hierarchical element within the low hierarchical level.

The first step of the projection process is to search for the most communicating pair of nodes in the ACG as shown in Figure 6. Subsequently, the most communicating pair of nodes is merged if the two nodes are hierarchically compatible. To be compatible, nodes must be potentially embedded in the same hierarchical element. It means that the hierarchical element must have enough available resources (functional elements) to implement the processing or memory operation described by the two corresponding nodes. If nodes are compatible, a new node called "composite node" is created to describe the merging of the nodes. Since the ACG has a

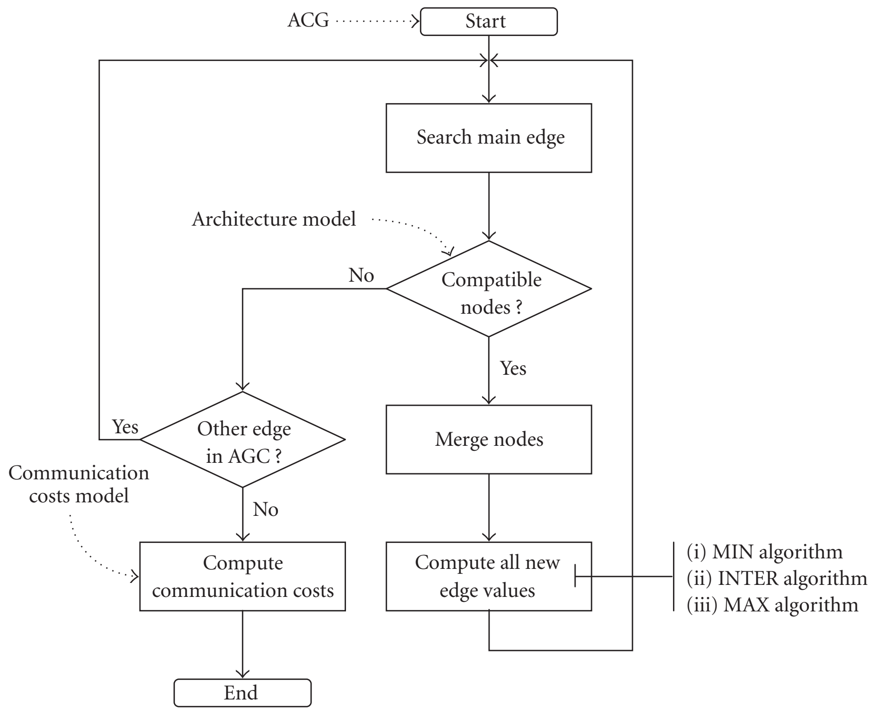


Figure 6: Architectural projection flow based on three algorithms: MIN, INTER, MAX.

new node, it is not the same graph, thus it is necessary to recompute all edge values and make several transformations due to the new composite node. The architectural projection stops when node merging is no longer possible and when all application resources are virtually associated to architecture functional elements. The complexity of the architectural projection algorithm is polynomial in $O(n^2)$, where n represents the number of edges within the ACG graph. This complexity does not represent a major issue as the number of edges is generally small. As explained previously, an edge represents the communications between two types of operations within the application.

Figure 7 shows the architectural projection process using the *INTER-algorithm*. For this example, the ACG and the architecture model are simple. The ACG has three processing nodes since the result of the system estimation tool allocated two multipliers, two subtracters, and one adder to respect the time constraint selected by the designer. The values on the ACG edges correspond to the number of required communications between the processing resources (corresponding to the processing nodes). For example, in Figure 7, twenty communications are required between the multipliers and the subtracters. The modeled architecture has two levels of hierarchy. In the hierarchical high level, one hierarchical element, H1, contains two elements H2. In the hierarchical low level, one hierarchical element, H2, contains three functional elements; two adders/subtracters and one multiplier.

In Figure 7, the process starts by merging the most communicating pair of nodes. The nodes multiplier and subtracter are the most communicating pair. As these two nodes are hierarchically compatible, in the second step a new composite node is created to describe that one subtracter and one multiplier are allocated in the same hierarchical element H2 in the architecture. The composite node has a number of internal communications; this number (ten in the case of Figure 7) depends on the algorithm (MIN, INTER, and MAX). In Figure 7, the process needs four steps to allocate all the required application resources in the modeled architecture. At the end, the ACG has two composite nodes; the number of communications in the low hierarchical level is computed. This number corresponds to the sum of the number of communications of the internal-composite nodes. For this example, there are 34.66 communications in the low hierarchical level (inside the hierarchical elements H2 between the functional elements) which corresponds to 82.5% of the total application communications. The next section will give the differences between the three algorithms.

5.3. Differences between the three architectural projection algorithms (MIN algorithm, INTER-algorithm, and MAX algorithm)

Three algorithms have been defined to merge the ACG nodes into composite nodes and to compute the ACG edge values for each architectural projection step. Figure 8 presents the first step of the hierarchical projection using the same example as Figure 7. The composite node is obtained from one subtracter and one multiplier merging.

The difference between the three algorithms is illustrated in Figure 8. The strategy of the *MIN algorithm* (top of Figure 8) is to consider that if two application resources (operator or memory) are assigned in the same hierarchical

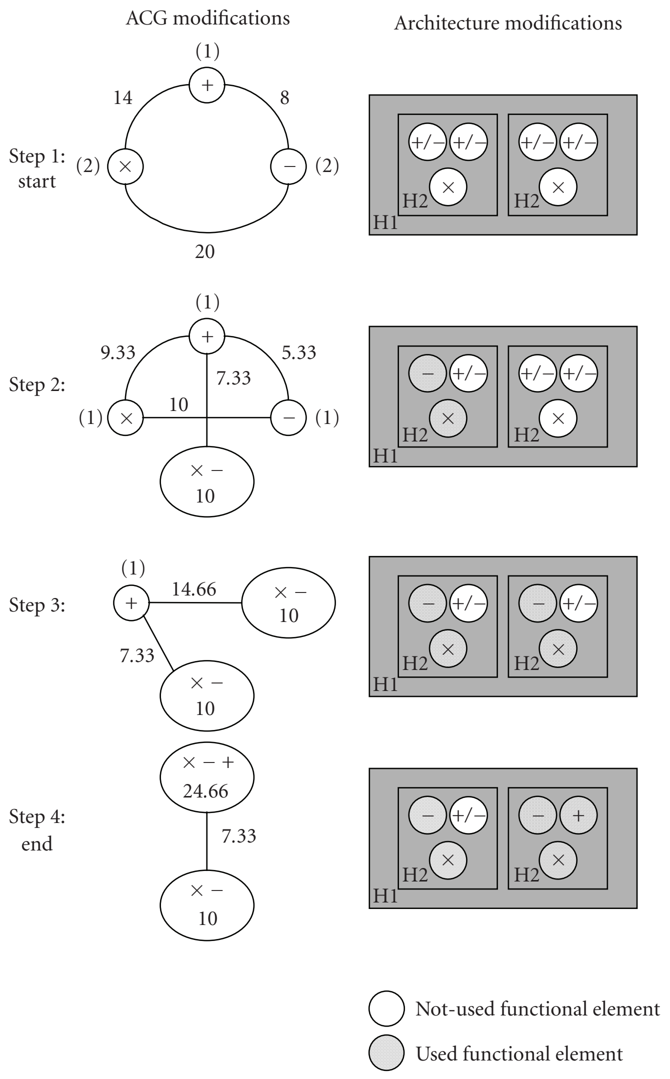


FIGURE 7: The architectural projection process, using inter algorithm with a simple three-node application ACG and two hierarchical-levels coarse-grain architectures.

element, all the communications between the two considered ACG nodes are allocated to the new composite node. Hence, when the composite node is created, the edge between the two initial nodes is deleted. Unlike *MIN algorithm*, the *MAX algorithm* does not take into account the creation of a new composite node, the communications between all the nodes and the composite node are distributed uniformly. In fact, the max algorithm corresponds to a greedy process to allocate the architectural resources. The idea of the *INTER-algorithm* is to consider that two operators in the same hierarchical element must communicate more than two operators in two different hierarchical elements; it is a tradeoff between min and max algorithms.

In order to have a better understanding of this approach, Figure 9 presents the three algorithms that estimate the communication costs. To understand the different algorithms, some notations are necessary:

(i) N_i shows node i of ACG,
(ii) t_i shows type of processing for the node N_i (processing such as adder, multiplier, etc.),
(iii) n_{ti} shows number of processings in the node N_i,
(iv) C_{ij} shows composite node with two processings t_i and t_j,
(v) $\mathrm{IC}_{C_{ij}}$ shows number of internal communications in the composite node C_{ij},
(vi) $E_{i,j}$ shows edge between nodes N_i and N_j,
(vii) $P_{i,j}$ number of communications between nodes N_i and N_j (i.e., value associated with edge $E_{i,j}$),
(viii) $E_{k,ij}$ shows edge between the node N_k and the composite node C_{ij},
(ix) $P_{k,ij}$ shows number of communications between the node N_k and the composite node C_{ij} (i.e., value associated with edge $E_{k,ij}$).

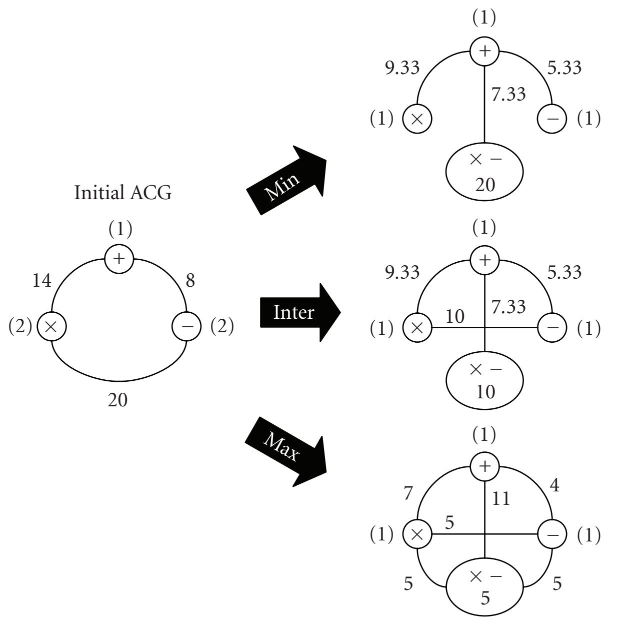


Figure 8: First step of the architectural projection for the three algorithms.

Some general functions are also used to describe the algorithms:

(i) CREATE_NEW_COMPOSITE(t_i, t_j) is the function that creates in the ACG a composite node with two processing types;

(ii) CREATE_EDGE(N_i, N_j) is the function that creates in the ACG an edge between nodes N_i and N_j;

(iii) DELETE_EDGE$(E_{i,j})$ is the function that deletes in the ACG the edge $E_{i,j}$ between nodes N_i and N_j;

(iv) MIN(*float1*, *float2*) is the function that returns the smaller float between *float1* and *float2*.

At the end of the architectural projection process, the designer obtains three communications hierarchical distribution estimations and the estimation of the architecture resources use rate. Based on these results, the designer can modify the architecture model according to the DSE method in order to reach his performance constraints. The designer goal is to define a power-efficient reconfigurable architecture for an application under a given time constraint. The next section presents the DSE method.

5.4. DSE method

Before explaining the exploration process, it is important to depict how an application is described since our method works at a high level of abstraction. The application is split into several functions and the execution of these functions can be either sequential or pipeline depending on the performances to achieve. The reconfigurable architecture must be efficient for all the functions of the application. Finding an optimal architecture for all the functions can be very tedious and even intractable. Moreover, searching for an optimal efficient architecture for all the functions independently is not the best way to quickly obtain the most efficient architecture for the total application. Several experiments [5] have shown that it is more efficient to identify the application critical functions and to perform the exploration for these functions since they have the strongest impact on the application performances. If the architecture is efficient for these functions, the application performances will be higher. To find these critical functions (often one or two functions within an application), we have developed three metrics that emphasize the realization characteristics of each function. These metrics are computed for each function.

(i) *The execution parallelism degree* of a function is obtained from the system estimation tool [19]. This metric highlights if the selected scheduling is suited for the function. As we target hardware reconfigurable devices, the parallelism degree must be high in order to benefit from the large amount of available resources. If the parallelism degree is low and if there is no other scheduling possibility, the function can be considered as critical since the designer has limited freedom to implement the function.

(ii) *The potential of communications spatial locality* of a function corresponds to the ratio between the number of resources (processing and memory) and the number of communications to be performed during the execution of the function. If there are many communications for a small number of resources, it is important to consider the spatial locality of communications since it will have a large impact on power efficiency. A function with such a feature can be considered as critical.

(iii) *The potential of architecture routing resources temporal congestion* during the execution of a function. This metric corresponds to the ratio between the number of function execution cycles (or time constraint) and the number of communications to be performed during the execution of the function. If there are many communications for a small execution time, it certainly will be challenging to temporally distribute the communications onto the routing resources. Therefore, the function can be considered as critical.

The critical functions are critical for all the metrics or for two among three. Usually there are one or two critical functions per application, but this number depends on the application complexity. The architectural exploration process is only performed for the application critical functions and leads to the definition of a power-efficient architecture that supports these functions under a given time constraint (scheduling choice, see Figure 2). If the application has several critical functions, then the final architecture corresponds to a tradeoff between the dedicated architecture for each function [16].

The definition of an efficient architecture for an application critical function begins with the analysis of its ACG (Figure 10). This analysis provides information about communications like the communications repartition between

Algorithm 1: Algorithm MIN

```
$C_{ij}$ = CREATE_NEW_COMPOSITE($t_i, t_j$)
$IC_{c_{ij}} = p_{i,j}$
for each $N_k \in$ ACG
/* compute edge value
between node k, node i, and composite node ij */
if ($N_k \neq N_i, N_k \neq N_j, N_k \neq C_{ij}$)
        if ($E_{k,i}$ exist)
                if ($E_{k,ij}$ exist)
                        $p_{k,ij} = p_{k,ij} + \frac{p_{k,i}}{n_{t_k} + n_{t_i}}$
                else
                        $E_{k,ij}$ = CREATE_EDGE($N_k, C_{ij}$)
                        $p_{k,ij} = \frac{p_{k,i}}{n_{t_k} + n_{t_i}}$
                end if
        end if
        $p_{k,i} = \frac{p_{k,i} \times (n_{t_k} + n_{t_i} - 1)}{n_{t_k} + n_{t_i}}$
end if
/* compute edge value
between node k, node j, and composite node ij */
/* idem that for node i but with node j */
...
end for
$n_{t_i} = n_{t_i} - 1$
$n_{t_j} = n_{t_j} - 1$
DELETE_EDGE($E_{i,j}$)
end
```

Algorithm 2: Algorithm INTER

```
$C_{ij}$ = CREATE_NEW_COMPOSITE($t_i, t_j$)
$IC_{c_{ij}} = \text{MIN}\left(\frac{p_{i,j}}{n_{t_i}}, \frac{p_{i,j}}{n_{t_j}}\right)$
if ($n_{ti} > n_{tj}$)
        $E_{i,ij}$ = CREATE_EDGE($N_i, C_{ij}$)
        $p_{i,ij} = \frac{p_{i,j}}{n_{t_i}} + \frac{p_{i,j}}{n_{t_j}}$
else if ($n_{ti} < n_{tj}$)
        $E_{j,ij}$ = CREATE_EDGE($N_j, C_{ij}$)
        $p_{j,ij} = \frac{p_{i,j}}{n_{t_j}} + \frac{p_{i,j}}{n_{t_i}}$
end if
for each $N_k \in$ ACG
/* compute edge value
between node k, node i, and composite node ij */
if ($N_k \neq N_i, N_k \neq N_j, N_k \neq C_{ij}$)
        if ($E_{k,i}$ exist)
                if ($E_{k,ij}$ exist)
                        $p_{k,ij} = p_{k,ij} + \frac{p_{k,i}}{n_{t_k} + n_{t_i}}$
                else
                        $E_{k,ij}$ = CREATE_EDGE($N_k, C_{ij}$)
                        $p_{k,ij} = \frac{p_{k,i}}{n_{t_k} + n_{t_i}}$
                end if
                $p_{k,i} = \frac{p_{k,i} \times (n_{t_k} + n_{t_i} - 1)}{n_{t_k} + n_{t_i}}$
        end if
end if
/* compute edge value
between node k, node j, and composite node ij */
/* idem that for node i but with node j */
...
end for
$p_{i,j} = p_{i,j} - IC_{c_{ij}} - p_{i,ij} - p_{j,ij}$
$n_{t_i} = n_{t_i} - 1$
$n_{t_j} = n_{t_j} - 1$
end
```

Algorithm 3: Algorithm MAX

```
$C_{ij}$ =CREATE_NEW_COMPOSITE($t_i, t_j$)
$IC_{c_{ij}} = \frac{p_{i,j}}{n_{t_i} \times n_{t_j}}$
$E_{i,ij}$ = CREATE_EDGE($N_i, C_{ij}$)
$p_{i,ij} = \frac{p_{i,j} \times (n_{t_i} - 1)}{n_{t_i} \times n_{t_j}}$
$E_{j,ij}$ = CREATE_EDGE($N_j, C_{ij}$)
$p_{j,ij} = \frac{p_{i,j} \times (n_{t_j} - 1)}{n_{t_i} \times n_{t_j}}$
for each $N_k \in$ ACG
/* compute edge value
between node k, node i, and composite node ij */
if ($N_k \neq N_i, N_k \neq N_j, N_k \neq C_{ij}$)
        if ($E_{k,i}$ exist)
                if ($E_{k,ij}$ exist)
                        $p_{k,ij} = p_{k,ij} + \frac{p_{k,i}}{n_{t_i}}$
                else
                        $E_{k,ij}$ = CREATE_EDGE($N_k, C_{ij}$)
                        $p_{k,ij} = \frac{p_{k,i}}{n_{t_i}}$
                end if
                $p_{k,i} = \frac{p_{k,i} \times (n_{t_i} - 1)}{n_{t_i}}$
        end if
end if
/* compute edge value
between node k, node j, and composite node ij */
/* idem that for node i but with node j */
...
end for
$p_{i,j} = \frac{p_{i,j} \times (n_{t_i} - 1)(n_{t_j} - 1)}{n_{t_i} \times n_{t_j}}$
$n_{t_i} = n_{t_i} - 1$
$n_{t_j} = n_{t_j} - 1$
end
```

FIGURE 9: Description of the three algorithms.

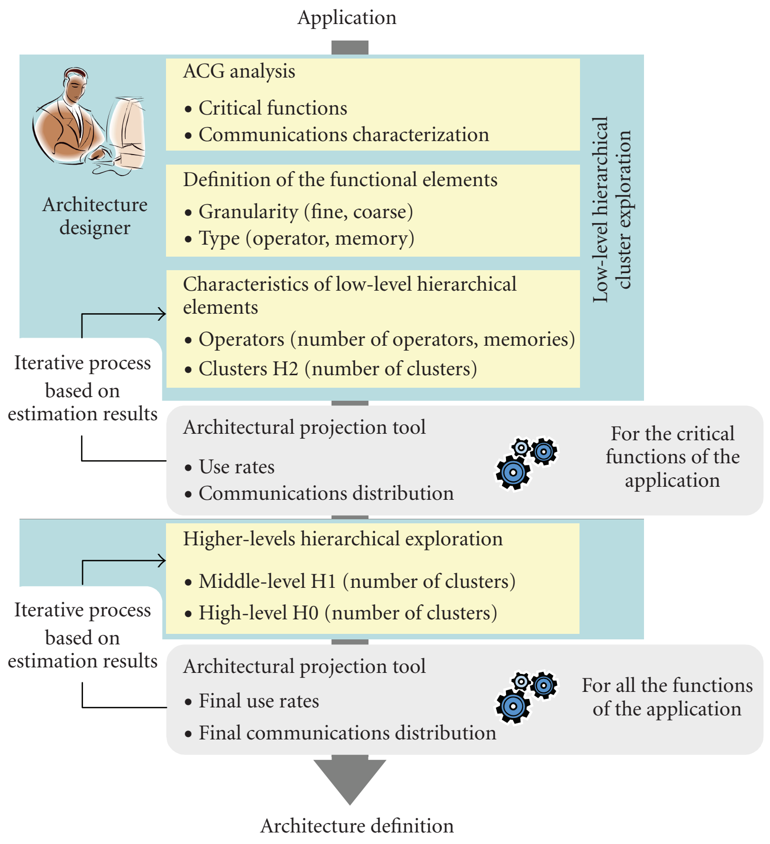

Application
ACG analysis
• Critical functions
• Communications characterization
Definition of the functional elements
• Granularity (fine, coarse)
• Type (operator, memory)
Characteristics of low-level hierarchical elements
• Operators (number of operators, memories)
• Clusters H2 (number of clusters)
Architecture designer
Low-level hierarchical cluster exploration
Iterative process based on estimation results
Architectural projection tool
• Use rates
• Communications distribution
For the critical functions of the application
Higher-levels hierarchical exploration
• Middle-level H1 (number of clusters)
• High-level H0 (number of clusters)
Iterative process based on estimation results
Architectural projection tool
• Final use rates
• Final communications distribution
For all the functions of the application
Architecture definition

FIGURE 10: Exploration flow for each critical function and then for the whole application to converge towards a power-efficient architecture.

the different computing and memory resources. The designer uses this information to build the architectural low-level hierarchical elements. At that level, the main issue corresponds to the definition of the granularity and the type of resources of the different functional elements (processing or memory) within the low hierarchical elements.

Then, the designer needs to determine the size of each low-level hierarchical element. For that purpose, the architectural projection tool is used to find the memory size and the number of each functional element embedded within the low-level hierarchical elements. To explore the architecture low level (memory size, number of functional elements), the designer manually changes the architecture model characteristics and launches the architectural projection tool. Designer modifications are based on the results provided by the previous architectural projection runs (architecture processing resources use rate and communications hierarchical distribution). It is an interactive and iterative process between the architectural projection tool results and the designer model modifications. As mentioned previously, we have developed an architectural model that enables a fast modification of the architecture characteristics. Furthermore, the architectural projection algorithm even if in $O(n^2)$ is very fast to compute a performance estimation as the number of edges is small. These two points are essential to mitigate the cost of the exploration process and to enable the designer to rapidly evaluate several architectures. Once a good size is found for each low-level hierarchical element, an exploration of the architecture higher hierarchical levels can be performed in order to complete the exploration process. When an efficient architecture is obtained for each critical function, a trade-off between all the architectures is define-based on the designer analysis of the solutions. These successive architectural exploration steps are highlighted in the next section, which gives the architectural exploration results for several applications for the image computing domain and the cryptographic domain.

6. APPLICATIONS

Four applications from the image processing and cryptography domains are considered to illustrate our exploration process. For each application, a power-efficient architecture has been targeted. The exploration process has mainly focused on the granularity of the processing and memory architectural resources. Each application has been specified in C language before being automatically translated into an HCDFG description.

This section is organized as follows: first, main characteristics of the considered image processing and cryptographic applications are given. The results of the system estimation tool for each application are presented before defining a power-efficient architecture for the two application domains.

Table 2: System estimation results.

Application	Comm.	Cycles	ADD/SUB	MUL/DIV	Comp.	Logic	Memory
ICAM	29.086.835	373.872.291	512	125	516	328	3.1 Mbytes
MPEG-2 encoder	40.745.280	45.476.864	398	279	153	33	60 Kbytes
Matching pursuit	3.751.397	239.215	232	162	69	0	6.3 Mbytes
AES core	1120	471	11	16	16	15	1 Kbytes

For each application and architecture, the architectural communications distribution estimates are given. The architectural resource use rate estimates are also provided in order to perform a whole analysis of the results.

6.1. *Image processing applications*

Three image processing applications have been used within our framework.

(i) *ICAM* (intelligent camera) is a motion estimation by intensity difference and reference background update [30]. This camera is used for subway supervision and crowd motion management in an urban environment.

(ii) *Matching pursuit* is an image compression application [31]. matching pursuit encoder is based on a genetic algorithm and can be implemented onto different platforms. Therefore, we work only on the decoder.

(iii) *MPEG-2* is a compression standard [32], which allows for the coding of studio quality video for digital TV, high-density CD-ROMs, and TV-broadcasting. We study only the encoder part of the MPEG-2 system.

6.2. *Cryptography application*

In order to not only confront our method to image processing applications, we have tested the method with a cryptography algorithm. We have chosen the last international advanced encryption standard AES.

AES algorithm has been developed to replace the DSE standard with a 128-bit key [33]. We have chosen the AES specification with 10 rounds and have not taken into account the key generator. We have focused on finding an efficient architecture for the cryptographic core.

6.3. *System estimation tool results*

The system estimation tool is used to perform the first step of the exploration process. Table 2 provides the selected results for each benchmark among the solutions provided by the tool. Table 2 design characteristics are the estimated number of communications within the application and the number of cycles to perform the application using the allocated number of processing and memory resources. The designer uses Table 2 to define an RTL logical architecture able to support the application. For that point, two ways are possible depending on the execution model: a sequential execution model which requires dynamic reconfiguration of the architecture or a pipeline execution model [5]. In the first case, the designer considers that the architecture is reconfigured with the adequate function at each step of the application execution. Hence, the reconfigurable architecture supports just one function at a time, in which case, each function can be implemented with a high degree of parallelism since a single function is running at a time. However, the dynamic reconfiguration consumes some time and power. We do not consider the dynamic reconfiguration process in our exploration (like previous efforts presented in Section 2). We consider the second execution model; pipeline execution. In that case, all the functions are implemented onto the architecture and during the whole execution time (any dynamic or partial reconfiguration). The function realization must be less parallel than in the first execution model due to area limitation. With this assumption, Table 2 shows that image computing applications are more resource consuming than cryptographic application. ICAM is the most complex application concerning the number of processing resources.

6.4. *Architectural exploration results*

Table 3 provides the results of the ACG study for the critical functions only. This study provides the average percentage of communications in the ACG between the following:

(i) the processing coarse-grain resources (intercoarse grain, Table 3-column3),

(ii) the processing coarse-grain resources and the processing fine-grain resources (coarse grain/fine grain, Table 3-column4),

(iii) the processing coarse-grain resources and the memory resources (coarse grain/memory, Table 3-column5),

(iv) the processing fine-grain resources (interfine grain, Table 3-column6),

(v) the processing fine-grain resources and the memory resources (fine grain/memory, Table 3-column7),

(vi) the memory resources (intermemory, Table 3-column8).

According to the flow presented in Figure 10, these results guide the designer to define a low-level cluster and particularly to identify if it is necessary or not to mix fine-grain and coarse-grain processing resources within the low-level clusters. These results show that it is efficient to build an architecture with separate fine-grain and coarse-grain clusters for the three first applications in order to have the maximum of fine grain processing resources in the same low-level hierarchical element (and the same thing for the coarse-grain processing resources). However, it is not the case for the last application, the AES core, since there is a large part of communications between coarse-grain and fine-grain resources and no intercommunications.

Table 3: Critical function ACG communication characterization.

Application	Number of critical functions	Intercoarse grain	Coarse-grain Fine grain	Coarse-grain memory	Interfine grain	Fine grain memory	Inter-memory
ICAM	2	2, 9%	0, 2%	15, 0%	19, 9%	36, 9%	25, 1%
MPEG-2 encoder	2	66, 9%	0, 7%	31, 1%	1, 1%	0, 2%	—
Matching pursuit	1	92, 3%	0, 1%	7.6%	—	—	—
AES core	1	—	15, 7%	28, 9%	—	25, 0%	30, 4%

According to these results, the designer can define four clusters that correspond to the atomic clusters of the final architecture for each application. Figure 11 provides a schematic representation of the four clusters. In this figure, the number of each functional element (processing or memory) is not relevant since this number is defined later in the exploration process.

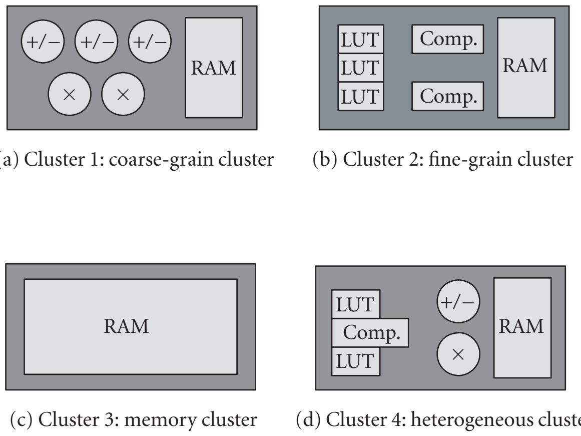
(i) Cluster 1 has two coarse-grain processing functional element types, adder/subtracter and multiplier, and one memory functional element. It is a coarse-grain cluster (Figure 11(a)).
(ii) Cluster 2 has two fine-grain processing functional element types, comparator and lookup table, and one memory functional element. It is a fine-grain cluster (Figure 11(b)).
(iii) Cluster 3 only has one large memory functional element, often used to store a complete picture in the case of image computing application. It is a memory cluster (Figure 11(c)).
(iv) Cluster 4 has four processing functional element types, adder/subtracter, multiplier, comparator and lookup table, and one memory functional element. It is a heterogeneous cluster (Figure 11(d)).

The architectural exploration leads to define the following:

(i) the number of processing functional elements for each type of functional element embedded in the low-level hierarchical cluster,
(ii) the size of the memory functional elements embedded in the low-level hierarchical cluster,
(iii) the number of low-level hierarchical clusters in the middle level hierarchical cluster,
(iv) and the number of middle-level hierarchical clusters in the high-level hierarchical cluster.

To perform the architectural exploration, the exploration rules based on the model hypothesis have to be satisfied; the communication costs inside a hierarchical element are homogeneous and the communications have less impact on the power consumption for low level of hierarchy than for high level of hierarchy.

Table 4 provides the number of processing functional elements embedded in each cluster (cluster 1, cluster 2, cluster 3, and cluster 4) for each application. According to the ACG study, cluster 4 is only used for the cryptography application. The image computing applications use the three other clusters. Concerning the ICAM application, two lines of Table 5 give the exploration results for two architectures (archi1 and archi2). The main difference between the two architectures is the cluster size. We use two different architectures in order to demonstrate that it is possible to obtain very good results with a nonrealistic architecture (archi2). This point will be discussed in the following section. Table 4 has to be jointly considered with Table 5 that gives the number of low-level hierarchical clusters. ICAM and MPEG-2 applications are the most complex applications, so they need more low-level clusters and larger clusters than the other two applications. AES application is less complex, so the cluster used for this application is smaller.



(a) Cluster 1: coarse-grain cluster (b) Cluster 2: fine-grain cluster (c) Cluster 3: memory cluster (d) Cluster 4: heterogeneous cluster

Figure 11: The four potential low-level clusters for the applications. Cluster 1, Cluster 2, and Cluster 3 are used for the three image processing applications. Cluster 4 is only used for the cryptographic core.

Once the designer has defined the low-level hierarchical clusters' size and number, he explores the middle level, and the high level of the hierarchy. The middle level embeds hierarchical elements like cluster 1, cluster 2, cluster 3, and cluster 4 (only for the AES application). The number of each cluster type in the middle-level for each application is given from row 2 to row 5 in Table 6. Row 6 provides the number of middle-level hierarchical elements embedded only in the high-level element.

Tables 4, 5, and 6 define the architectural exploration results for each application (with two possible architectures for ICAM application). These results provide the designer with an estimation of processing functional elements use rate and an estimation of the communications hierarchical distribution. The following section details these estimations.

Table 4: Processing functional element number embedded in the low hierarchical level cluster.

Application	Number of ADD/SUD in cluster 1	Number of MUL in cluster 1	Number of COMP in cluster 2	Number of LUT in cluster 2	Number of ADD/SUD in cluster 3	Number of MUL in cluster 3	Number of COMP in cluster 3	Number of LUT in cluster 3
ICAM archi1	4	1	5	2	—	—	—	—
ICAM archi2	20	10	21	13	—	—	—	—
MPEG-2 encoder	4	4	2	1	—	—	—	—
Matching pursuit	8	6	3	0	—	—	—	—
AES core	—	—	—	—	1	1	1	1

Table 5: Number of low hierarchical level clusters.

Application	Number of cluster 1	Number of cluster 2	Number of cluster 3	Number of cluster 4
ICAM archi1	130	234	26	0
ICAM archi2	26	26	26	0
MPEG-2 encoder	105	105	0	0
Matching pursuit	30	25	5	0
AES core	0	0	0	16

6.5. *Estimation results*

Table 7 provides the use rate estimations of each type of processing functional element for each application. The designer targets the highest use rate because unused resources reduce the power efficiency, particularly for coarse-grain processing resources. However, the problem is more complex because often the designer must choose a tradeoff between use rate and communications distribution (highest number of communications in the architecture hierarchical low level). For example, we have defined an architecture with a very high use rate for the matching pursuit application (Table 7 line 4) and an architecture with a lower use rate for MPEG-2 decoder application (Table 7 line 5). The issue is now to analyze the communications hierarchical distribution, since it also has a significant impact on the final performances. Table 8 provides the communications hierarchical distribution estimation. The number of communications estimated in the low level is higher for the MPEG-2 decoder application than for the matching pursuit application. Moreover, the number of communications estimated in the high level of hierarchy is lower for the MPEG-2 decoder application than for the matching pursuit application. The communications hierarchical distribution is better for the MPEG-2 decoder than for the matching pursuit application, but as we have seen, this is not the case for the use rate. Hence, the designer must choose the best solution in terms of tradeoff between use rate and communications hierarchical distribution according to the technological process used for his architecture.

To provide a schematic representation of the architecture dedicated for the MPEG-2 decoder, Figure 12 gives a representation of the three hierarchical levels: high level (Figure 12(a)), middle level (Figure 12(b)), and low level (Figure 12(c)).

Concerning the ICAM application, Table 8 shows that the estimation results obtained with the archi2 are better than with the archi1. Nevertheless, as seen in Table 5, the low-level clusters are five times larger on the average. With such an archi2 large cluster, it is difficult for the designer to find a solution that guarantees that the communication cost is homogeneous within the cluster. Therefore, the archi2 is not a realistic architecture except if a communication technology enables providing homogeneous cost within the cluster in terms of delay and power.

Concerning the AES application, it uses another type of architecture than the image computing applications. The architecture for the AES application has only one low-level cluster type with fine-grain and coarse processing functional element (heterogeneous cluster). As for the MPEG-2 dedicated architecture, Figure 13 presents a schematic representation of the three hierarchical levels of the architecture: high level (Figure 13(a)), middle level (Figure 13(b)), and low level (Figure 13(c)). Table 8 shows that the estimation results of the communication distribution are very good for this application with 69% of the communications in the architectural low level and only 21% in the high level. However, before concluding that this method leads to define an efficient architecture for several application domains, it is important to estimate the communications distribution with the architecture highlighted for image computing where the coarse-grain and fine-grain processing functional elements are separated. The last line in Table 8 provides the estimation results in this case (AES core ic-archi). The number of communications in the architectural low level is reduced by 19% and the number of communications in the architectural high level is increased by 15%. Therefore, the architecture defined for the image computing applications in not adapted for the cryptography application. It shows that according to the discussion in Section 1.2, this method enables the definition of dedicated reconfigurable architectures for different application domains.

7. CONCLUSION

Design space exploration for reconfigurable architectures combined with algorithmic exploration of applications is an important issue which has been insufficiently addressed till now. We propose in this paper an original approach based

Table 6: Exploration results of middle and high hierarchical levels.

Application	Middle-level hierarchical element				High-level hierarchical element
	Number of cluster 1	Number of cluster 2	Number of cluster 3	Number of cluster 4	
ICAM archi1	5	9	1	0	26
ICAM archi2	2	2	2	0	13
MPEG-2 encoder	7	7	0	0	15
Matching pursuit	6	5	1	0	5
AES core	0	0	0	4	4

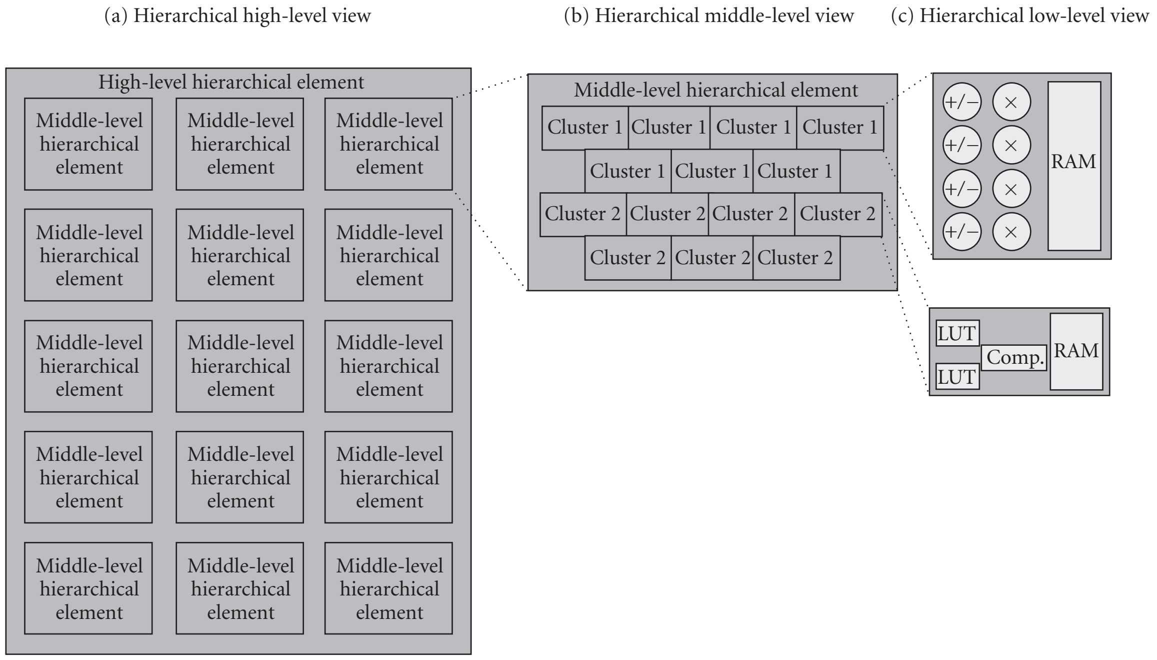

(a) Hierarchical high-level view
(b) Hierarchical middle-level view
(c) Hierarchical low-level view
High-level hierarchical element
Middle-level hierarchical element
Middle-level hierarchical element
Cluster 1
Cluster 2
+/−
×
RAM
LUT
Comp.

Figure 12: Schematic representation of special reconfigurable architecture for the MPEG-2 application.

Table 7: Use rate estimation of each processing functional element type.

Application	ADD/SUB	MUL	COMP	LUT
ICAM archi1	98, 5%	96, 1%	36, 7%	70, 1%
ICAM archi1	98, 5%	48, 0%	94, 5%	97, 0%
MPEG-2 encoder	67, 0%	70, 0%	13, 0%	2, 0%
Matching pursuit	97, 0%	90, 0%	92, 0%	—
AES core	63, 8%	100%	100%	93, 8%

Table 8: Hierarchical distribution communication estimation.

Application	High level	Middle level	Low level
ICAM archi1	28%	35%	37%
ICAM archi2	13%	30%	57%
MPEG-2 encoder	29%	8%	63%
Matching pursuit	31%	32%	37%
AES core	21%	10%	69%
AES core ic_archi	36%	14%	50%

on a high-level representation of the application and on a hierarchical functional model for the architecture. Our approach targets fine-grain, coarse-grain, and heterogeneous architectures.

To perform the exploration of the architecture space, two metrics have been defined, the architectural processing use rate and the communications hierarchical distribution since we have shown (particularly with fine-grain architecture studies) that these metrics are significant in reducing the power consumption of an application under a given time constraint. The exploration process leads to the definition of a power-efficient hierarchical reconfigurable architecture for an application or an applications family. We have demonstrated the efficiency of our approach for image processing and cryptography applications. In order to provide the designers with estimates of the achievable performances, we have defined an estimation technique that computes an interval of performance. This point is important and more relevant than an optimal estimation technique considering the level of abstraction of our approach. The goal is to greatly

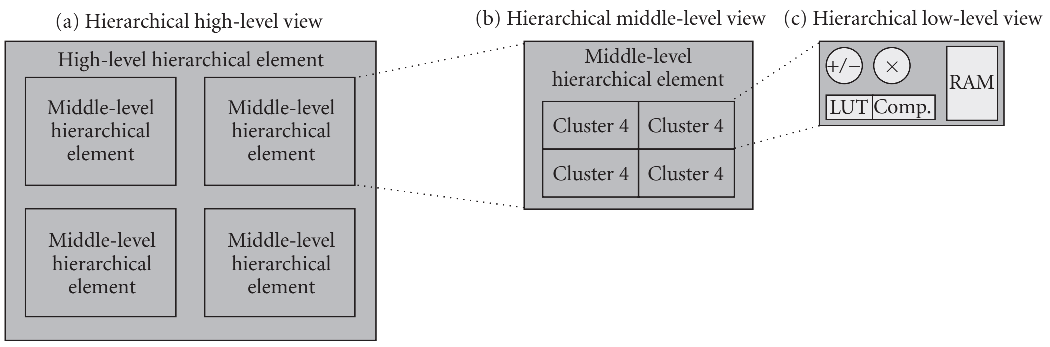

(a) Hierarchical high-level view
High-level hierarchical element
Middle-level hierarchical element
Middle-level hierarchical element
Middle-level hierarchical element
Middle-level hierarchical element
(b) Hierarchical middle-level view
Middle-level hierarchical element
Cluster 4
Cluster 4
Cluster 4
Cluster 4
(c) Hierarchical low-level view
+/−
×
LUT
Comp.
RAM

FIGURE 13: Schematic representation of dedicated reconfigurable architecture for the AES application.

prune the design space in order to shorten the design cycle and to rapidly converge towards the definition of a power-efficient reconfigurable architecture. The estimation results demonstrate that our approach rapidly leads to defining a power-efficient architecture for an applications domain. This point is essential since it is a current trend to specialize the reconfigurable architectures for a specific domain.

REFERENCES

[1] N. Tredennick and B. Shimamoto, "The rise of reconfigurable systems," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '03)*, pp. 3–12, Las Vegas, Nev, USA, June 2003.

[2] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," in *Proceedings of Conference and Exhibition on Design, Automation and Test in Europe (DATE '01)*, pp. 642–649, Munich, Germany, March 2001.

[3] P. Schaumont, I. Verbauwhede, K. Keutzer, and M. Sarrafzadeh, "A quick safari through the reconfiguration jungle," in *Proceedings of the 38th Design Automation Conference (DAC '01)*, pp. 172–177, Las Vegas, Nev, USA, June 2001.

[4] A. D. Pimentel, L. O. Hertzberger, P. Lieverse, P. van der Wolf, and Ed. F. Deprettere, "Exploring embedded-systems architectures with artemis," *Computer*, vol. 34, no. 11, pp. 57–63, 2001.

[5] L. Bossuet, *Exploration de l'espace de conception des architectures reconfigurables*, Ph.D. thesis, Université de Bretagne Sud, Vannes, France, September 2004.

[6] M. Gries, "Methods for evaluating covering the design space during early design development," Technical Memorandum MO3/32, Electronics Research Laboratory, University of California, Berkeley, Calif, USA, August 2003.

[7] V. Betz and J. Rose, "VPR: a new packing, placement and routing tool for FPGA research," in *Proceedings of the 7th International Workshop on Field Programmable Logic (FPL '97)*, pp. 213–222, Oxford, UK, September 1997.

[8] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," in *Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '00)*, pp. 3–12, Moterey, Calif, USA, February 2000.

[9] S. J. E. Wilton, J. Rose, and Z. G. Vranesic, "The memory/logic interface in FPGA's with large embedded memory arrays," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 80–91, 1999.

[10] L. Lagadec, *Abstraction, modélisation et outils de CAO pour les circuits intégrés reconfigurables*, Ph.D. thesis, Université de Rennes1, Rennes, France, 2000.

[11] S. Choi, J. W. Jang, S. Mohanty, and V. K. Prasanna, "Domain-specific modeling for rapid system-level energy estimation of reconfigurable architectures," in *Proceedings of International Conference of Engineering of Reconfigurable Systems and Algorithms (ERSA '02)*, Las Vegas, Nev, USA, June 2002.

[12] R. Enzler, T. Jeger, D. Cottet, and G. Tröster, "High-level area and performance estimation of hardware building blocks on FPGAs," in *Proceedings of the the Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications (FPL '00)*, pp. 525–534, Villach, Austria, August 2000.

[13] C. A. Moritz, D. Yeung, and A. Agarwal, "Exploring optimal cost-performance designs for Raw microprocessors," in *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '98)*, pp. 12–27, Napa Valley, Calif, USA, April 1998.

[14] U. Nadelginder, *Coarse-grain reconfigurable architecture design space architecture exploration*, Ph.D. thesis, University of Kaiserslautern, Kaiserslautern, Germany, June 2001.

[15] R. Kress, *A fast reconfigurable ALU for xputers*, Ph.D. thesis, University of Kaiserslautern, Kaiserslautern, Germany, 1996.

[16] L. Bossuet, G. Gogniat, and J.-L. Philippe, "Generic design space exploration for reconfigurable architectures," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS '05)*, p. 163, Denver, Colo, USA, April 2005.

[17] Design Trotter Project: http://web.univ-ubs.fr/lester/~diguet/Design-TrotterPage.html.

[18] J. P. Diguet, G. Gogniat, P. Danielo, M. Auguin, and J.-L. Philippe, "The SPF model," in *Proceedings of Forum on Design Language (FDL '00)*, Tübingen, Germany, September 2000.

[19] Y. Le Moullec, P. Koch, J. P. Diguet, and J.-L. Philippe, "Design trotter: building and selecting architectures for embedded multimedia applications," in *Proceedings of IEEE International Symposium on Consumer Electronics (ISCE '03)*, Sydney, Australia, December 2003.

[20] Y. Le Moullec, J. P. Diguet, T. Gourdeaux, and J.-L. Philippe, "Design trotter: system-level dynamic estimation task a 1st step towards platform architecture selection," *Journal of Embedded Computing*, vol. 1, no. 4, pp. 565–586, 2005.

[21] L. Bossuet, G. Gogniat, and J.-L. Philippe, "Fast design space exploration method for reconfigurable architectures," in

Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '03), pp. 65–71, Las Vegas, Nev, USA, June 2003.

[22] T. Mudge, "Power: a first-class architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52–58, 2001.

[23] S. Rouxel, "Caractérisation de l'impact du routage sur les performances (vitesse et consommation de puissance) d'un FPGA," M.S. thesis, Université de Bretagne Sud, Lorient, France, September 2003.

[24] D. Elleouet, "Caractérisation et modélisation de la consommation de puissance des mémoires sur FPGA," M.S. thesis, Université de Bretagne Sud, Lorient, France, September 2003.

[25] A. Garcia, W. Burleson, and J.-L. Danger, "Power modelling in field programmable gate arrays (FPGA)," in *Proceeding of the 9th International Workshop on Field Programmable Logic and Applications (FPL '99)*, pp. 396–404, Glasgow, Scotland, August-September 1999.

[26] V. George, H. Zhang, and J. Rabaey, "The design of a low energy FPGA," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '99)*, pp. 188–193, San Diego, Calif, USA, August 1999.

[27] E. Kusse and J. M. Rabaey, "Low-energy embedded FPGA structures," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '98)*, pp. 155–160, Monterey, Calif, USA, August 1998.

[28] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic power consumption in virtex^{TM}-II FPGA family," in *Proceedings of the 10th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '02)*, pp. 157–164, Monterey, Calif, USA, February 2002.

[29] K. K. W. Poon, A. Yan, and S. J. E. Wilton, "A flexible power model for FPGAs," in *Proceeding of the 12th International Conference on Field-Programmable Logic and Applications (FPL '02)*, pp. 312–321, Montpellier, France, September 2002.

[30] H. Zhang, M. Wan, V. George, and J. Rabaey, "Interconnect architecture exploration for low-energy reconfigurable single-chip DSPs," in *Proceedings of the IEEE Computer Society Workshop on VLSI (WVLSI '99)*, p. 2, Orlando, Fla, USA, April 1999.

[31] CEA. Intelligent Camera—3D Methodology. http://www-list.cea.fr/fr/programmes/systemes_embarques/docs/ICAM_internet_list_v0.pdf.

[32] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3397–3415, 1993.

[33] MPEG2, http://www.mpeg2.de.