Communication costs driven design space exploration for reconfigurable architectures

Lilian Bossuet, Guy Gogniat, Jean-Luc Philippe

LESTER Lab
University of South Brittany,
Lorient, France
{lilian.bossuet, guy.gogniat, jean-luc.philippe}@univ-ubs.fr

Abstract

In this paper we propose a design space exploration method targeting reconfigurable architectures that takes place at the algorithmic level and aims to rapidly highlight architectures that present good performance vs. flexibility tradeoffs. The exploration flow is based on a functional model to describe the architectures that the designer wants to compare. The paper mainly focuses on the projection step of our flow and presents an allocation heuristic that is based on communication costs reduction.

1. Introduction

The new telecommunication and multimedia applications need to have a reduction of actual systems on chip (SoC) power consumption and an increase of SoC flexibility. Although the evolutions of integration and design are more and more important, they are not sufficient to face these challenges. It is necessary to provide novel approaches that work at the system level to design more efficiently, and to target new technologies.

Reconfigurable architectures are becoming more attractive in terms of capacity, performances, low-power consumption and flexibility (through the possibilities of run-time reconfiguration and multi-granularity resources) [1][2]. They correspond to an efficient solution to the SoC challenge and will be unavoidable in a near future. But the design space of reconfigurable architectures is very large, because these architectures can be extremely heterogeneous in term of processing, memory and routing resources. Hence, it is very complex to find the best reconfigurable architecture for a panel of applications where each application can be dynamically configured on the architecture.

In order to help the designer it is necessary to develop tools that compare several architectures for different applications. We propose in this paper an original method of design space exploration for reconfigurable architectures that works at the algorithmic level.

The paper is organized as follows. Section 2 describes related work dealing with design space exploration methodologies. Section 3 describes major issues in design space exploration at the algorithmic level and section 4 presents our approach. Section 5 details the projection step. Section 6 gives some results for different reconfigurable architectures. Finally, section 7 concludes the paper and exposes future direction.

2. Related work

Many research teams are focusing on reconfigurable architecture [1]. Some are working on design space exploration methodology in order to find the best architecture for a panel of applications.

Two ways are possible to explore reconfigurable architectures. The first one is to synthesize all the applications for the different target architectures, and to compare the overall performance results. In that case the results are very accurate, but it is necessary to have a specific synthesis tool for each architecture (which is not always available in the case of architecture exploration) or to use generic synthesis tools [3][4]. However, synthesis steps use very complex algorithms, which conducts to a limited and slow exploration. Furthermore it is necessary to have a very good knowledge of the target architectures when using generic synthesis tools since it is necessary to provide them a model of the target architectures. Hence, this method is not really adapted for a large and rapid architecture exploration and is more dedicated to do some architecture refinement steps.

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

Both exploration methods require having an architecture model. It is possible to consider a physical model. Then it is necessary to know precisely the physical parameters of the architecture (technology, routing type and size, routing switch resources, clusters size, etc). Versatile Place and Route (VPR) tool, developed at the Toronto University, is a very interesting approach that works on a physical model [3]. VPR is a synthesis tool that works at the logic level and is oriented for island style fine-grained architectures (as FPGA). It is not suitable for coarse-grained architectures. It is also possible to model architecture with a functional model. Each element of the architecture is described by the functions it can execute. The functional model enables to describe a large panel of architectures and the description are technological independent. This model is used in the generic place and route tool for fine-grained reconfigurable architectures called Madeo-Bet [4] that works at the logic level.

VPR and Madeot-Bet are not the only tools that use an architecture model, but there are very representative. Both are FPGAs oriented, but other approaches target reconfigurable architectures. In [5] the design space exploration flow targets mesh architecture called KressArray - a fast reconfigurable ALU. The exploration tool, Xplorer works at the algorithmic level and aims to assist the designer in finding 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.

[6] presents the design space exploration for the Raw Microprocessor as an example of a tiled architecture. The Raw Microprocessor is reminiscent of coarse-grained FPGA and comprises a replicated set of tiles coupled together by a set of compiler orchestrated, pipelined, switches. Each tile contains a RISC-like processing core and SRAM memory for instructions and data.

3. Design space exploration flow principles

Design space exploration can be performed at different levels of abstraction in order to reduce progressively the number of solutions (figure 1). More the abstraction level is refined more accurate results can be obtained since a lower number of solutions need to be considered.

At the algorithmic level the objective is to rapidly identify target architectures that present a high performance and versatility potential. To reach such a goal, design space exploration methods must promote the flexibility, the rapidity and the fidelity. Encouraging (i) the flexibility means that performance and versatility can be estimated for a wide variety of reconfigurable architectures, (ii) the rapidity enables to estimate performances without the time-consuming computation of programs such as Place & Route algorithms and (iii) the fidelity points out that the relative comparisons between two alternative architectures must be close to the relative errors that would be obtained after the synthesis steps even if there may be significant absolute errors in the performance estimation at the algorithmic level.

Another major concern is to promote the interactivity with the designer at all the abstraction levels in order to take benefit from its experience. Thus, the refinement process at a given abstraction level can be performed through several runs of the exploration method in order to converge progressively to an efficient mapping between the application and the architecture. Between several runs (two runs for the example of the figure 2), the designer can improve the architecture model according to the previous results of the exploration. Once, the designer has selected some efficient architectures he can refine its results by decreasing the abstraction level and thus using more accurate architecture model and exploration tools.

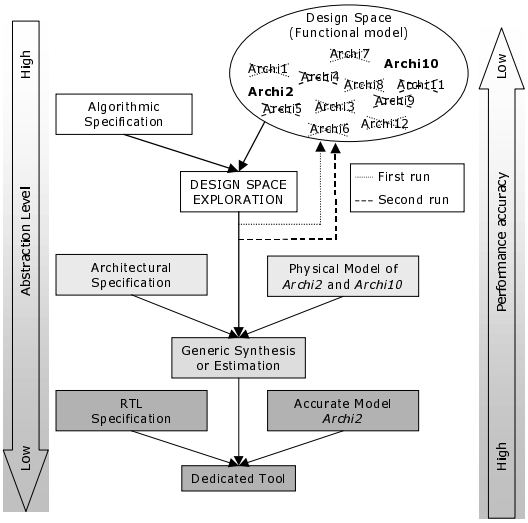


Figure 1. Design Space Exploration

In this paper we proposed a design space exploration method targeting reconfigurable architectures that addresses the previous highlighted principles. Our method is based on an estimation approach and takes place at the very first steps of the design flow since it works at the algorithmic level. A functional model is used to describe the target architectures since such model as proven its efficiency to characterize a wide variety of reconfigurable architectures [7].

4. Proposition of a design space exploration flow

The design space exploration flow that we propose is depicted figure 2. The specification is provided in a high level language (C language) and is first translated into an intermediate representation - the HCDFG model. This model is a Hierarchical Control and Data Flow Graph allowing efficient algorithm characterization and

exploration of complex applications including control flow and multi-dimensional data.

The first part of the flow (figure 2) is the **System Estimation** step [8][9], during which the application is characterized and scheduled. The results computed are defined as **Costs Profiles** i.e., scheduling for all the resources used by the application and for different time constraints. The available processing and memory resources to perform the scheduling are defined in a file called **User Abstract Rules**.

Relative Estimation, the second step of the flow (figure 2), aims to estimate the application performances on several reconfigurable architectures. Relative Estimation gives designer information to improve progressively the architecture definition with several runs in the design exploration flow.

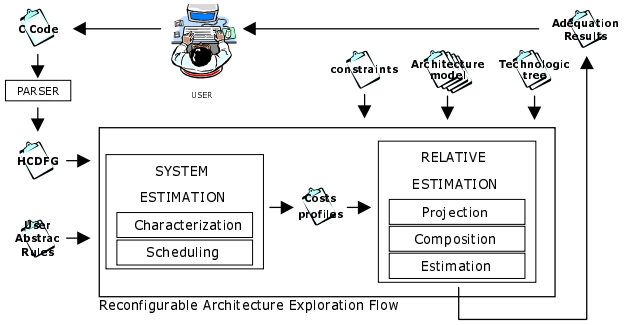


Figure 2. Design Exploration Flow

4.1. Specification

The application is specified with the C language. Once it has been functionally validated, it is translated into an intermediate model, which is a Hierarchical Control and Data Flow Graph (HCDFG) [10]. For sake of simplicity we can say that an application is modeled with several DFG connected through control structures, hierarchy and dependence relations. An important characteristic of the HCDFG model is that processing and data are explicitly represented with nodes in the graph. This decomposition enables to highlight the hierarchy and the potential parallelism of the application, which is an essential characteristic to perform an efficient algorithmic and architecture exploration.

4.2. System Estimation

System Estimation [8] consists in two steps; **Characterization** and **Scheduling**. During the Characterization step, the application orientation is analyzed through three axes - processing, control and memory. Specific metrics are used to find this orientation [9]. Once the application orientation has been exhibited the scheduling of the application is performed accordingly. For example, if the application is processing oriented, the processing resources are first scheduled and the memory resources are scheduled in a second step (the opposite if the application is memory oriented). The main objective of the System Estimation is to highlight the intrinsic processing and memory parallelisms of the application and to give some guidance on how to build the architecture (pipeline, parallelism, memory hierarchy etc.). Further details on the system estimation step are beyond the scope of this paper. Interested reader can refer to [9].

4.3. Relative Estimation

Relative Estimation is linked with the System Estimation through the cost profiles. The cost profiles describe the scheduling results (for all the processing and memory operations) for different time constraints. These values characterize the application to be implemented.

This step is composed of three tasks: **projection**, **composition** and **estimation** that are performed sequentially. In order to evaluate an application on different architectures, it is necessary to specify the target reconfigurable architectures. For this, a functional model is used since such model is suitable for rapid relative comparison and has proven its efficiency to describe architectures at a high abstraction level. This model called **HF model** [7], enables to describe functionally the elements of the architecture and to represent different architectural styles and different architecture elements. Although, the routing resources are not explicitly described, they are taken into account in the estimation flow. Connection resources are taken into account by connection costs in the HF model. There are two types of elements in the model: the hierarchical elements and the functional elements. The hierarchical elements are used to describe the architecture hierarchy. A hierarchical element can be composed of functional elements and other hierarchical elements. The functional elements are used to describe the architecture resources. They can be logical, input/output, memory and processing resources. An extension of the HF model exists to model tile-based architectures, where the communications between the tiles must be explicitly modeled [11]. Figure 3 shows an example of architecture with several levels of granularity

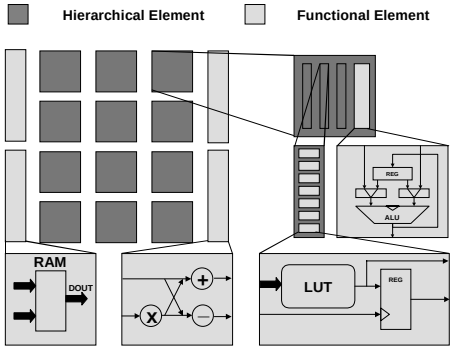


Figure 3. Example of reconfigurable architecture could be specified with the HF

that can be modeled with the HF model.

Relative Estimation begins with the **Projection** step that makes the link between the functional needs (processing and memory) of the application and the available resources of the architecture. If a difference of resource granularity exist between the functional needs (from the application) and the available resources (in the architecture), the necessary resources can be split as fine-grained resources using a library called **Technological Trees** (the same method is used with the PipeRench reconfigurable architecture [12]).

The **Composition** step takes into account the application scheduling obtained during the System Estimation step in order to refine previous results. Since it is necessary to add resources dedicated to realize the scheduling like multiplexer, register or states machine. These additional resources are taken into account in the last step of the estimation process.

The **Estimation** step computes the global application performances on the selected architectures. The estimations take into account the static costs of the model (interconnect costs between two hierarchical elements and the costs of the functional elements), and the dynamic costs of the application (critical path, operator communications, memory reads/writes). The results of this step are gathered into a file where the application is characterized for the target architectures.

Composition and estimation steps are not the topic of this article.

5. Projection Algorithms

In this section we focus on the projection step which is particularly important in the flow since it has a strong impact on the quality of the final estimated performances. The goal of this step is to allocate the resources of the architecture that will support the application operations (processing and memory). The allocation algorithms aim to assign in a same hierarchical level of the architecture the resources that communicate the most in order to reduce the cost overhead due to communications. In our model architecture hierarchy corresponds to routing topology. Communication costs are smaller inside a hierarchical element than between two hierarchical elements. More hierarchical levels are crossed by a communication higher is the communication cost.

Why focussing on communication costs for allocation algorithms? Studies on power repartition in fine-grain reconfigurable architecture like FPGA show that the major contribution to power consumption is due to routing resources (wires and switch). It is always better to reduce communication paths [13][14] since implementations are more energy efficient when wires are short and number of switches is low. The clock frequency can also be increased when communication paths are short since critical path is reduced.

In our approach we enhance the spatial locality between resources that most communicate. Different cost functions have been defined, to estimate the communication costs, which are computed in three algorithms that give respectively a lower and an upper bound and a mean value for the total communication costs.

Since our approach works at the algorithmic level it does not target a specific synthesis tool and does not consider an accurate physical architecture model. Hence instead of giving designers a single communication cost value that may present a significant absolute error due to backend synthesis algorithms and architecture refinement we propose to give them some bounds and an average value. Such approach as shown figure 4 enables designers to select architectures at the algorithmic level with the guaranty that the final performance will belong to the estimated performance interval.

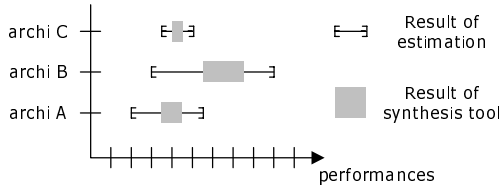


Figure 4. Bounded performances results

Such approach also enables to give designer metrics on allocation algorithm impact. On the example (figure 4) the architecture C has a narrow performance interval so allocation algorithms will have a small effect on the final performance and low complexity algorithms can be considered. The architecture B has a large performance interval so allocation heuristics will have a strong impact on final performance and it might be important to consider better allocation algorithms.

Results of the relative estimation step provide designers resources utilization rate and estimations of communication costs. Based on these results designers can perform a first architectures performance comparison and remove architectures that present a poor synergy with the application (e.g. unadapted granularity) or too important communication costs.

5.1. Average Communications Graph (*ACG*)

In order to make this projection, the first step builds, from the HCDFG model, a new graph with only processing nodes, i.e. a graph without memory node and control structure. The edges between two nodes represent the communications. This graph is then reduced in order to take into account the scheduling result, the final graph is called Average Communications Graph (ACG).

The ACG exhibits how each type of processing resources communicates with the other types of processing resources. This graph is used during the projection step to enhance the spatial locality of

communicating resources. In the ACG each edge represents the communications between two types of processing nodes.

Figure 5 shows on a simple example how to transform a processing graph in an ACG. In the ACG, each node corresponds to a type of processing resources. In our example the type of the processing resources correspond to a letter (a, b, c or d). Several differences exist between the two graphs. The ACG is realized after the processing graph scheduling. There are fewer nodes in the ACG than in the processing graph, since the ACG graph has only one node for one type of processing resource. The ACG edges are not oriented, the communications are take into account in all directions. Several attributes are added in the ACG to better describe the communications inter-node.

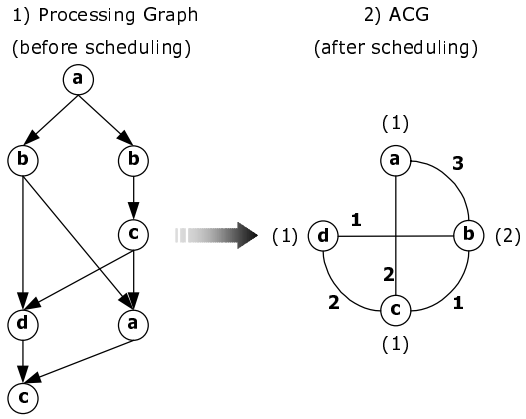


Figure 5. Transformation of the processing graph (a) into ACG (b)

The number in brackets beside a node is the number of operators that the scheduling has allocated (see section 4.3). The boldface number beside an edge is the total number of communications between two processing types. In order to know what pair of nodes communicates the most, it is necessary to compute the relative number of communications between two processing types. This value is obtained with the following expression:

$$RelativeComm_{Op1-Op2} = \frac{TotalComm_{Op1-Op2}}{NumberOp1 + NumberOp2}$$

Where $RelativeComm_{Op1\text{-}Op2}$ is the relative number of communications, $TotalComm_{Op1\text{-}Op2}$ is the total number of communications, *NumberOp1* and *NumberOp2* are the numbers of allocated operator of each type. For example, the ACG on the figure 5 has two nodes **a** and **b** linked by an edge with the value equal to 3. The node **a** describes one operator with the **a** type and the node **b** describes two operators with the **b** type. So the relative number of the edge, between this two nodes, is given by:

$$RelativeComm_{a-b} = \frac{3}{1+2} = 1$$

It is very fast to build the ACG from the HCDFG of the application. The ACG is the input of the projection algorithms.

5.2. Generic projection algorithm

The projection step makes the link between the necessary (application) and the available (architecture) resources with the challenge that the most communicating resources must be assigned in a same hierarchical level of the architecture. Figure 6 shows the proposed algorithm. The algorithm begins with the ACG, and the first step searches in the graph the pair of nodes that communicates the most. This step searches the edge with the highest relative value.

When a pair of nodes is determined it is necessary to know if the two types of processing node can be implemented by functional elements in a same hierarchical element.

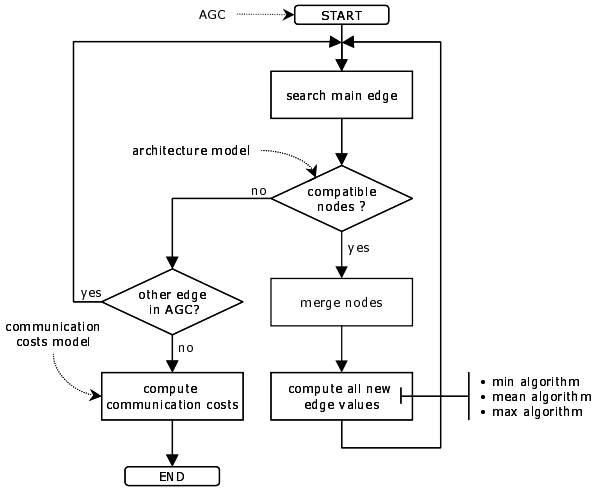


Figure 6. Generic projection algorithm

If the two nodes are compatible, they are assigned to the hierarchical element, and they form a new node, a composite node. This composite node has as parameter the processing types of the two previous nodes. Since the ACG has a new node, it is not the same graph, so it is necessary to re-computed all this edge values and make all the necessary transformations due to the composite edge presence. To do this, we use three algorithms that give respectively a lower and an upper bound and a mean value for the total communication number. These algorithms are detailed in the paragraphs 5.3, 5.4 and 5.5. Each of them gives the total number of communications in the architecture to support the application.

If, after the search of the main edge, the pair of nodes is not compatible, the search re-starts with other nodes. If all the ACG edges have already been selected, and if any pair of nodes is compatible (it is not possible to implement it in functional elements in a same hierarchical element), then the projection algorithm stops.

To compute the communication costs a communication costs model that is based on the architecture hierarchy must be considered. A communication between two hierarchical elements does

not represent the same cost than a communication in a single hierarchical element. More details are given in the section 5.6.

In the following we present the three algorithms that estimate the communication costs during the allocation step. To understand the different algorithms, some notations are necessary:

- N_i: Node *i* of ACG
- t_i: Type of processing for the node N_i (processing such as adder, multiplier, …)
- n_{ti}: Number of processings in the node N_i
- C_{ij}: Composite node with two processings t_i and t_j
- IC_{Cij}: Number of internal communications in the composite node C_{ij}
- $E_{i,j}$: Edge between nodes N_i and N_j
- $P_{i,j}$: Number of communications between nodes N_i and N_j (i.e. value associated with edge $E_{i,j}$)
- $E_{k,ij}$: Edge between the node N_k and the composite node C_{ij}
- $P_{k,ij}$: Number of communications between the node N_k and the composite node C_{ij} (i.e. value associated with edge $E_{k,ij}$)

Some general functions are also used to describe the algorithms:

- CREATE_NEW_COMPOSITE(t_i, t_j): Function that creates in the ACG a composite node with two processing types.
- CREATE_EDGE(N_i, N_j): Function that creates in the ACG an edge between nodes N_i and N_j.
- DELETE_EDGE($E_{i,j}$): Function that deletes in the ACG the edge $E_{i,j}$ between nodes N_i and N_j.
- MIN(*float1*, *float2*): Function that returns the smaller float between *float1* and *float2*.

5.3. Min algorithm (lower bound)

Figure 7 gives the algorithm that computes a lower bound of the communications number. To illustrate the execution of the min algorithm, figure 8 shows on a very simple example the different steps of computation. The modifications of the architecture are presented on the right side of the figure. The architecture has two hierarchical levels (represented by two hierarchical elements H1 and H2). The hierarchical element H2 has three functional elements, one functional element can realize one multiplication and the two other functional elements can realize one addition or one subtraction (depends on the configuration). During the process the functional elements are progressively allocated (in grey). On the left side of the figure the modifications of the ACG during the process are detailed. At the beginning of the process, the ACG has no composite node. At each step one composite node is created by merging two nodes, and all the ACG edge values are re-compute to take into account the new composite node.

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

Figure 7. Min Algorithm

The strategy of the min algorithm is to consider that if two operators of distinct processing types can be assigned in a same hierarchical element, they will perform all the communications between all the nodes of both processing types. It is the reason why in the start of the algorithm no edge is created between the new composite node (that describe a hierarchical realization) and one of the two nodes that communicate the most. For the example of the figure 8, the node multiplier and the node subtracter shape the pair of nodes that communicate the most (the relative value of the edge between this pair of node is the higher of the ACG). So in the second step of the algorithm, a composite node is created with two operations; one multiplication and one subtraction. The numbers (in brackets) of allocated operator multiplier and subtracter are decremented. The edge between the pair of nodes is deleted but this value (20) is transformed into internal communications of the new composite node.

However if another node has some communications (i.e. an edge) with one of the two previous nodes, then this node has an edge with the new composite node, and it preserves its other edges but with new values. It is the case of the node adder in the example of the figure 8.

The process stops when it is not possible to merge nodes. When the process ends, the communication cost is computed. In this example, the cost to communicate between two hierarchical elements H2 (in the H1

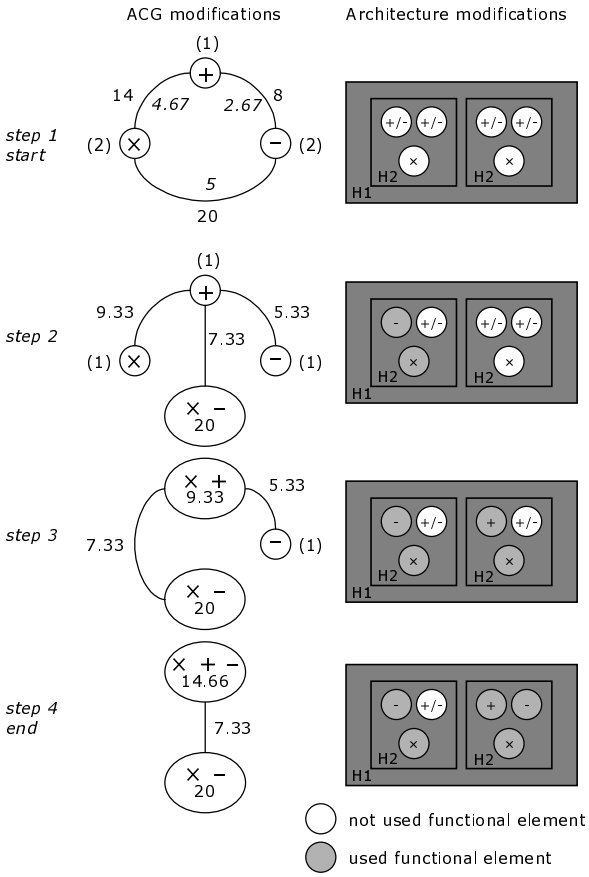


Figure 8. Example of min algorithm processing

element) is 0.2, and the cost to communicate in a H2 element is 0.1.

With this algorithm most communications are executed in composite nodes. As the communication costs in composite nodes are low (because there is not communication across several hierarchical levels) the total cost of communication will represent a lower bound.

5.4. Max algorithm (upper bound)

Figure 9 gives the algorithm that determines an upper bound of communication number. As in the case of the min algorithm, figure 10 shows on the same example the different steps of computation. The strategy of the max algorithm is to consider that all operators communicate uniformly. When a composite node is created with a pair of nodes (that communicates the most in the ACG) two edges are created between the new composite node and the pair of nodes. We can see the creation of composite node in the second step of the figure 10. The communications are uniformly distributed between the two edges value and the internal communications in the new composite node.

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. Max Algorithm

With this algorithm the communications are uniformly executed between the different hierarchical elements, which corresponds to an upper bound. In that case the allocation algorithm do not take benefit from the architecture hierarchy, hence, the communication costs correspond to maximum costs.

5.5. Mean algorithm (mean value)

Figure 11 gives the last algorithm that computes a mean value of communication number by a hierarchical projection. As in the case of the min algorithm, figure 12 shows the same example the different steps of computation.

The idea of the mean algorithm is to consider that two operators in a same hierarchical element must communicate more than two operators in two different hierarchical elements. The number of communication in a

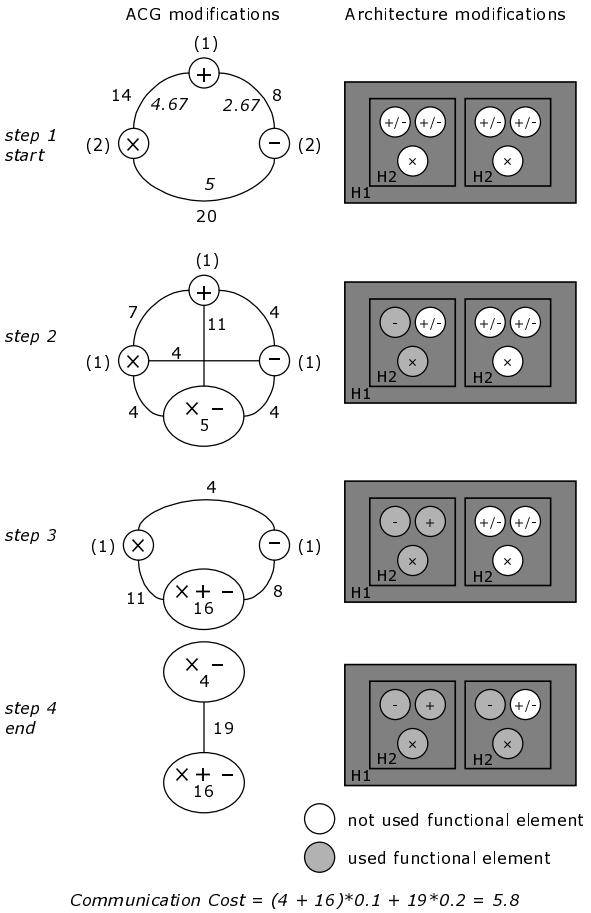


Figure 10. Example of max algorithm processing

composite node, is the maximum communication value between two nodes. That is the reason why the MIN function is used to determine the internal communications in the new composite node.

We can see on the figure 12 that contrary to the min and max algorithms, with the mean algorithms, the two first created composite nodes are with one multiplier and one subtracter, that is most appropriated to this application. But as this example is very simple, when the communication costs are compute, the result is the same with the mean that with the min algorithm. But with a bigger application it is not the case as we can see in the section 6.

C_{ij} = CREATE_NEW_COMPOSITE(t_i ,t_j)

$$IC_{c_{ij}} = MIN\left(\frac{p_{i,j}}{n_{t_i}}, \frac{p_{i,j}}{n_{t_j}}\right)$$

if ($n_{ti} > n_{tj}$)

$E_{i,ij}$ = CREATE_EDGE(N_i, C_{ij})

$$p_{i,ij} = \frac{p_{i,j}}{n_{t_i}} + \frac{p_{i,j}}{n_{t_j}}$$

elsif ($n_{ti} < n_{tj}$)

$E_{j,ij}$ = 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

$$p_{i,j} = p_{i,j} - IC_{c_{ij}} - p_{i,ij} - p_{j,ij}$$

$$n_{t_i} = n_{t_i} - 1$$

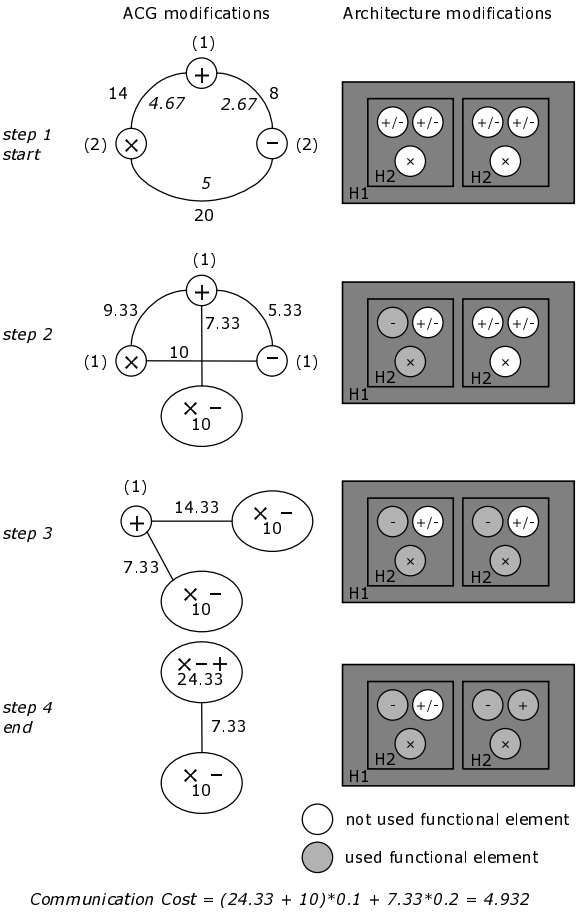
$$n_{t_j} = n_{t_j} - 1$$

end

Figure 11. Mean Algorithm

5.6. Communication costs

On figure 6, the final step of the projection algorithm uses a communication costs model to compute the total communication cost. This model describes the costs to perform a data transfer in a hierarchical element and between several hierarchical elements. These costs depend on the routing topologies (mesh, segmented base, hierarchical, etc) and on the routing resources (channel of wires, bus, crossbar, etc). A survey of interconnects architectures for reconfigurable architecture is done in [15]. In our approach these costs are relative since we want to compare two architecture styles instead of giving an absolute performance value. However, if the designer has a good knowledge of the target architecture, precisely costs can be considered.



*Communication Cost = (24.33 + 10)*0.1 + 7.33*0.2 = 4.932*

Figure 12. Example of mean algorithm processing

Once the communication costs are determined, it is necessary to compute the number of communication between each hierarchical element in function of the graph result (result of one of the three previous algorithms). The total communication cost is given by the following expression:

$$CommCost = \sum_{i}^{n} \left(\sum_{j}^{m_i} C_j \right)_i \times k_i$$

Where there are n hierarchical elements in the target architecture. The total number of communication in the element i is the sum of communication C_j inside each of the m_i hierarchical element i. The cost for one communication in the hierarchical element i is k_i. In this expression the fisrt sum with the i index allow to scan all the architecture hierarchical elements. The second index j allow to sum all the communication inside each hierarchical element i.

6. Application and results

The one dimension Lee-DCT, a typical application of video compression [16], has been chosen to exhibit the exploration process and its ability to compare several architectures. Figure 13 presents the processing graph and

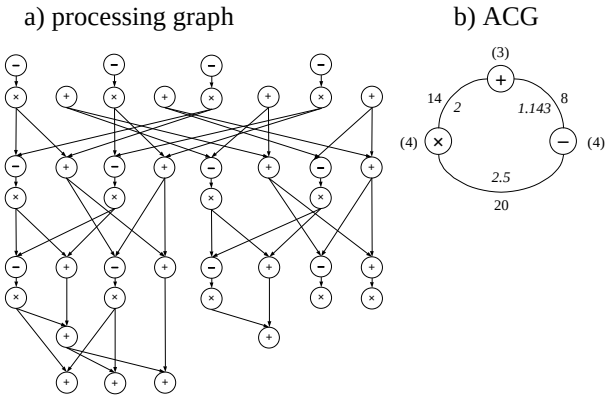


Figure 13. Processing graph, and ACG for the 1-D Lee DCT

the ACG for this application. The processing scheduling during the system estimation step has given four multipliers, four subtracters and three adders to perform this application.

Four architectures have been targeted during the projection step, with different hierarchical levels, and cluster sizes (number of functional elements for a hierarchical element). Figure 14 depicts it.

The results of the projection step is the total number of communications between resources weighted by the costs to communicate between hierarchical elements (communication costs model). For the considered architectures the communication costs model is the following (for the example of architecture **A**):

- Cost of communication internal H2 = 0.1
- Cost of communication H2 - H2 = 0.2
- Cost of communication H2 - H1 = 0.3

Results are given in table 1. They highlight several interesting elements. Architectures **A** and **B** are better for this application than architectures **C** and **D**, since in the application the base structure is a MAC so it is necessary

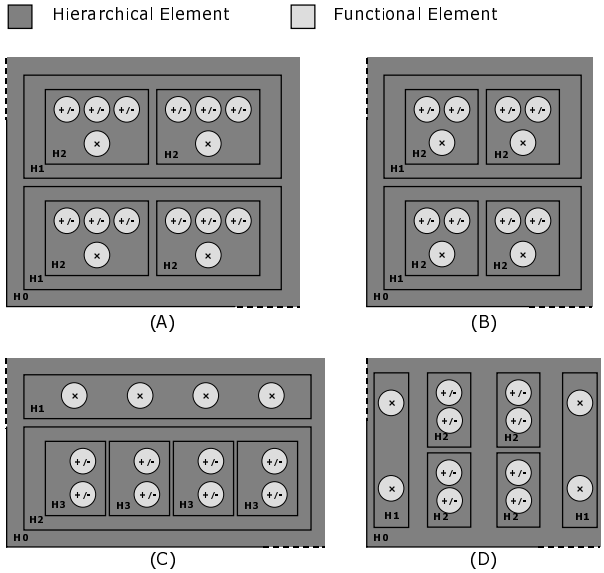


Figure 14. Four examples of target architectures

to have in a same hierarchical element the possibility to assign one multiplier, one adder and one subtracter.

We can notice that the results are close between the MIN and the MAX algorithm for the architecture **C** and **D**. Since these architectures have a limited exploration potential for this application, so the allocation will not have a strong impact on the final performances.

The last row of the table 1 gives the utilization rate of functional elements in the architecture. Although the **A** architecture is better to reduce communication costs, it has not a high functional element utilization rate. The **A** architecture is larger that the other architectures, so its static power consumption is higher than the other architectures. With the utilization rate information, we can see that **B** architecture is a good tradeoff between communication cost reduction vs. functional element utilization rate.

architecture	A	B	C	D
MIN	4.5	4.5	11	7.6
MEAN	6	6.5	11.3	7.9
MAX	9.2	9.45	11.6	8.2
USE	68.7%	91.6%	91.6%	91.6%

Table 1. Results of projection algorithms

In order to show the ability of our approach to make architecture exploration, we have computed the mean communication cost for architectures **A** and **B** with several numbers of hierarchical elements H2 in the hierarchical H1. We can see, on figure 15, that the larger is the cluster H1, the better is the communication cost. Indeed if H1 has many H2, local communications are promoted. This experiment show how is easy to explore some architecture characteristics.

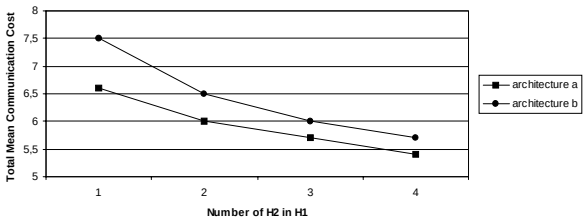


Figure 15. Communication cost versus H1 size

7 Conclusions and future work

In this paper we have presented a rapid allocation algorithm that takes into account the communications between application's operations. This algorithm uses the hierarchical view of the target architecture that is modeled with a functional model. Communication costs are taken into account with a communication cost model that depends on the routing structure in the target architecture. This algorithm is part of a design exploration flow that works at the algorithmic level and enables to compare quickly several reconfigurable architectures.

These approach which has been integrated in the codesign environment *Design Trotter* [8], enables to explore a large design space at an early stage of the design cycle and to characterize each solution in terms of area versus delay.

Future work will complete the design exploration flow development and test it on a tile-based architecture [11].

8 References

[1] R. Hartenstein. A Decade of Reconfigurable Computing : a Visionary Retrospective. *In DATE'01, Munich, Germany, 13-16 March, 2001.*

[2] J. M. Rabaey. Reconfigurable Processing: The Solution to Low-Power Programmable DSP. *In IEEE, ICASSP'97, Munich, Germany, April 1997.*

[3] V. Betz, J. Rose, A. Marquart. *Architecture and CAD for Deep Submicron FPGAs.* Kluwer Academic Publishers, 1999.

[4] L. Lagadec, D. Lavenier, E. Fabiani, B. Pottier. Placing, Routing and Editing Virtual FPGAs. *In FPL'01, August 2001.*

[5] U. Nageldinger. Coarse-Grained Reconfigurable Architecture Design Space exploration. *Ph.D. Thesis, University of Kaiserlautern, Germany, June 2001.*

[6] C. A. Moritz, D. Yeung, A. Agarwal. Exploring Optimal Cost-Performance designs for Raw Microprocessors. *In FCCM'98, Napa, CA, USA, April 1998.*

[7] L. Bossuet, G. Gogniat, J.P. Diguet, J.L. Philippe. A Modeling Method for Reconfigurable Architecture. *In IWSOC'02, Banff, Canada, July, 2002.*

[8] Y. Le Moullec, J.P. Diguet, J.L. Philippe. Design Trotter: a Multimedia Embedded Systems Design Space Exploration Tool. *In IEEE MMSP'02, Virgin Island, USA, 9-11 December, 2002.*

[9] Y. Le Moullec, N.Ben Amor, J.P. Diguet, M. Abid, J.L. Philippe. Multi-Granularity Metrics for the Era of Strongly Personalized SOCs. *In DATE 03, Munich, Germany, 3-7 March, 2003.*

[10] J. P. Diguet, G. Gogniat, P. Danielo , M. Auguin, J.L Philippe. The SPF model. *In FDL'00, Tübingen, Germany, September, 2000.*

[11] L. Bossuet, W. Burleson, G. Gogniat, V. Anand, A. Laffely, J.L. Philippe. Targeting Tiled Architectures in Design Exploration. *RAW'03, Nice, France, April, 2003.*

[12] M. Budiu, S. C. Goldstein. Fast Compilation for PipeRench Reconfigurable fabrics. *In ACM/SIGDA FPGA'99, Monterey, CA, USA, February, 1999.*

[13] L. Shang, A. Kaviani, K. Bahala. Dynamic Power in VirtexTM-II FPGA Family. *In ACM/SIGDA FPGA'02, Monterey, Californie, USA, February, 2002.*

[14] A. Garcia, W. Burleson, J.L. Danger. Power Modeling in Field Programmable Gate Arrays (FPGA). *In FPL'99, Glasgow, Scotland, September 1999.*

[15] H. Zhang, M. Wan, V. George, J. Rabaey. Interconnect Architecture Exploration for Low-Energy Reconfigurable Single-Chip DSPs. *IEEE Computer Society Workshop on VLSI, April 1999.*

[16] V. Bhaskaran, K. Konstantinides. *Image and Video Compression Standards : Algorithms and Architectures*. Kluwer Academic Publishers, 1995.