Algorithmic-level Specification and Characterization of Embedded Multimedia Applications with Design Trotter

Yannick Le Moullec, Jean-Philippe Diguet, Nader Ben Amor,
Thierry Gourdeaux, Jean-Luc Philippe
LESTER, Université de Bretagne Sud, 56321 Lorient Cedex, France
Jean-Philippe.Diguet@univ-ubs.fr

Abstract

Designing embedded system is a non-trivial task during which wrong choices can lead to extremely costly re-design loops. The extra cost is even higher when wrong choices are made during the algorithm specification and mapping over the selected architecture. In this paper we propose a high-level approach for design space exploration, based on a usual standard language. More precisely we present the characterization step which is located at the very beginning of our hardware / software codesign framework. Once transformed into our internal representation, the specification is rapidly and automatically characterized and explored at the algorithmic level. Metrics are provided to the designer in order to evaluate very early in the design process, the impact of algorithmic choices on resources requirements by means of processing, control, memory bandwidth capabilities and potential parallelism at different levels of granularity. The aim is to improve the algorithm / architecture matching that sorely influences the implementation efficiency in terms of silicon area, performances and energy consumption. We give examples which illustrate how designers can refer to the outcomes of our methodology in order to select or build suitable architectures for specific applications.

Keywords: specification, characterization, algorithm / architecture matching, design-space exploration, SoCs.

1 Introduction

The context of our work is the hardware/software co-design in the domain of embedded multimedia applications. In this area, algorithmic and architectural choices have a strong impact on the power vs. performance trade-off which is a key issue regarding the evolution of mobile electronic devices. Thus, designers have to face a number of challenges. Three main situations can be considered: i) a chosen target architecture must be used, in that case optimizations have to be carried out on the specification and its implementation; ii) the specification cannot be changed, in that case optimizations have to be performed on the architecture which has to be selected

and/or tailored to match the specification and iii) neither the specification nor the architecture are fixed, in that case optimizations have to be performed using feedbacks between them. In all cases the designer needs relevant knowledge about the specification. Such knowledge includes operation granularity, potential parallelism, orientation (processing, control, memory) and data locality (memory hierarchy).

Consequently, a fast automated exploration process is required to alleviate the designer with the tedious task consisting in evaluating a large number of potential solutions based on the previously mentioned algorithmic options.

We tackle this problem by considering a high-level algorithmic approach using a standard procedural language for specifying the application. This specification is then automatically transformed into a fully graph-based representation which enables a fast and automatic characterization and exploration of the application in terms of algorithmic options.

Namely we consider a system as event-based at the highest levels of hierarchy - Hierarchical Finite State Machines (HFSMs) or task-graphs (TG)- encapsulating function calls. These functions are described with Hierarchical and Control Data-flow Graphs (HCDFGs), presented in this paper. It is the responsibility of the designer to choose the granularity of the specification, and therefore his responsibility to choose what should be described by means of HFSMs/task-graphs and HCDFGs. However, since this task is not trivial the designer can use the characterization step (presented in section 4) to get metrics about the control or data-flow orientation of the functions and iterate towards the most appropriate separation. It is worth noting that tools are available for simulation, formal proof and code generation at the event-based level. The goal of our work is to performed automatically and rapidly the tedious algorithmic exploration for the functions called from the event-based level.

This work is part of a complete design framework called Design Trotter, presented in [1]. This is a set of cooperative tools which aim at guiding embedded system designers early in the flow by means of design space exploration, as summarized in Fig. 1. It operates at a high-level of abstraction (algorithmic-level). Firstly the different functions of the applications are handled separately. For each fonction, the first step presented in this paper, includes the graph building (HCDFG specification) and the characterization metric computation. Then a scheduling step [1] is performed for each data-flow graph and loop nest within the function, the design space exploration is obtained by means of a large set of time constraints (e.g. from the critical path to a complete sequential execution). Finally, the results are combined to produce trade-off curves (#ressource vs. #cycles). The scheduling process offers different options including the balance between data-transfers and data-processings and the use of loop unrolling to meet time constraints. After the

intra-function scheduling an inter-function scheduling step [2], based on the previous trade-off curves, can be performed if functions can be executed concurrently. Then a projection step enables the exploration of the design space targeting reconfigurable architectures (FPGAs) [3], [4] and processors (HW and SW projections in Fig. 1 respectively). Finally results can be used within our HW/SW partitioning and real-time scheduling tool [5].

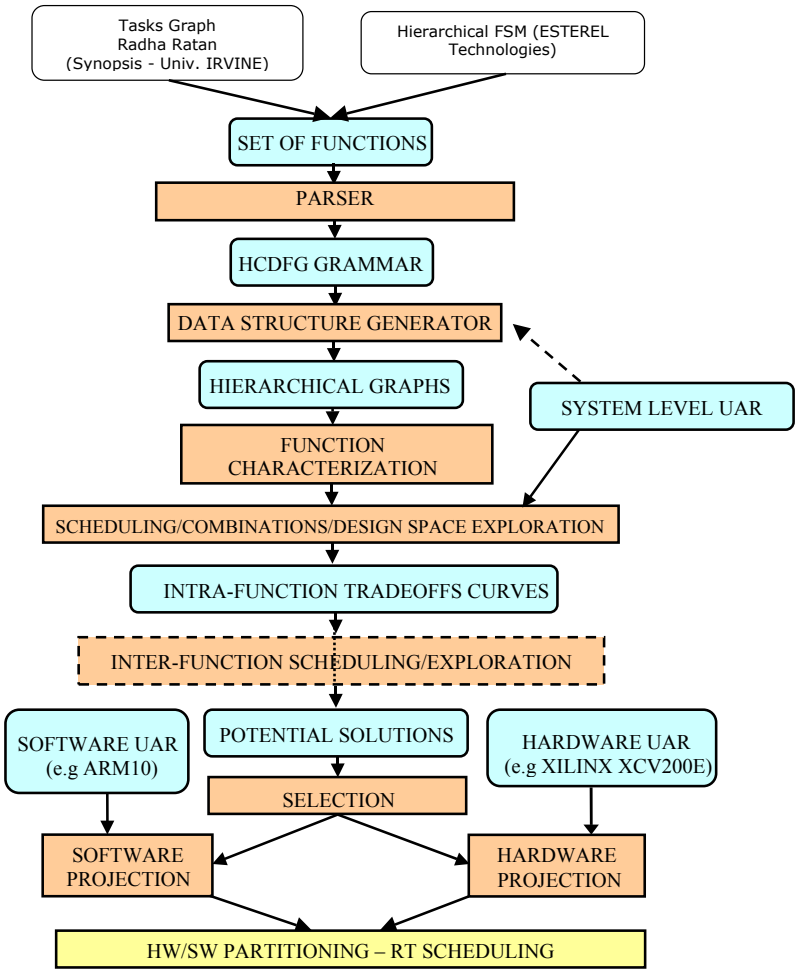

Figure 1: Design Trotter design flow.

The rest of the article is organized as follows: in section 2 we give an overview of existing specification models and characterization tools. Then, we expose the contributions of our paper for both specification and characterization aspects. Sections 3 and 4 detail these two points respectively. In section 5 we present some results of the characterization step showing the interest of the proposed method. Finally in section 6 we conclude about our work and present some perspectives.

2 Related work

2.1 Part 1: specification

One of the first issue when designing a system is its specification. Specifying a system is a complex task which can have a major influence on the subsequent steps of the design flow. Choosing a specification model can, for example, stress more or less hardware vs. software orientations, event-based vs. data-flow based approaches, data-processings vs. data-transfers. Granularity is another important feature which greatly influences the possibilities of the exploration process. For example a fine grain granularity specification focusing on implementation details usually enables very accurate results at the cost of long exploration processes. It is crucial to control these aspects and not to be restricted to one level of granularity (for subsequent steps such as characterization, cf. section 4).

2.1.1 Existing work for specification

Generally, existing design approaches consider only partial design flow. The decomposition of the design flow results from the architectural target and/or from the type of information handled by the application. We can mention control oriented models [6], [7],[8],[9], data-flow models [9],[10], Khan process networks [11], Ptolemy so-called domains (e.g. DE, DF).

To alleviate this problem, some attempts to define an unified modeling and design framework have been proposed. They are generally based on co-simulation (e.g., CoWare, VCC from Cadence). However, these approaches do not really support a complete and seamless design flow, from high-level specification to the generation of VHDL code for synthesis and assembly code for processors.

SystemC [12] is a design and verification language enabling the description of systems from high-levels (algorithmic-level) down to implementation in hardware and software. The modeling features of SystemC are provided as a C++ class library. It is a possible candidate for becoming a standard for the design of embedded systems. Reducing the productivity gap in system design can be achieved by raising the level of abstraction. However, as mentioned in [13], it is important to well-define the abstraction levels and models. The authors propose system-level semantics that cover the system design process and define properties and features for each model. By formalizing the flow, design automation for synthesis and verification can be performed and productivity gains can be achieved. Moreover, customizing the semantics enables the creation of specialized design methodologies.

2.1.2 Contribution

Initiatives like SystemC and SpecC [14] aim to provide the designer with a common language for progressively modeling an application by refining steps, from the system un-timed level to the implementation cycle accurate level. Such framework are necessary, however the designer remains responsible for choosing the appropriate computation model. One of the main influent specification decision is the separation between an event-based FSM model and a data-flow model, the boundary is not necessarily clear and can vary depending on the designer background.

The requirements of the tools included in Design Trotter have guided our choice towards a suitable internal representation model. This model should have the following features:

- Hierarchy: enables structured exploration of the specification such as bottom-top and top-bottom approaches but also partial exploration;
- Multi-granularity: enables coarse and/or fine grain specification depending on the utilized tool: fast characterization, system-level estimation (scheduling/combinations/exploration in Fig. 1), architectural estimations (HW and SW projections in Fig. 1);
- Attributes: used to save critical information. We consider two types of attributes: those created during the parsing step (e.g. data-type) and those created during the exploration/characterization steps (e.g. metrics);
- Parallelism specification: parallelism is a key feature in design space exploration. The model must enable the clear specification of processing and data-transfers parallelism in order to detect and exploit them;
- Openness: the model must be open to new languages and can be easily changed/extented.

We found out that none of the existing (and easily available) models had all the required features. Therefore we have defined our own model: HCDFG. The definition of its formalism (and underlying grammar rules) has been guided by a pragmatic approach considering that the most important point is to stay close to the original algorithmic specification of the application. This internal representation model, based on graphs, offers the flexibility required for the development of the tools implemented in Design Trotter.

We preferentially use the Esterel framework [15] to perform the first specification step since it offers various interesting features such as concurrent and hierarchical finite state machines (HFSM called Safe State Machine in Esterel framework) and tools for formal proof computation, simulation and code generation. Our approach enables a clear control of the model separation in a top down

approach: when the designer estimates that a data-flow specification can be efficiently used at a given state level of the HFSM he can insert calls to C functions. The HCDFG description starts at this level. Then depending on metric computation results (cf. 4), the designer can decide to modify his specification choices in terms of HFSM/C separation. We use the C language for three main reasons, firstly a lot of standards (e.g. ITU) and applications are written with this language, secondly it can be naturally inserted in various framework like SystemC and Esterel, and finally the GCC compiler provides a good starting point for building a graph representation. Actually, we use GCC to perform lexical and grammatical checkings and then introduce our own syntactic tree in order to extract the information we consider relevant (see 3.3).

In our approach the next step after the specification is the characterization of the application. In what follows we present existing work on this aspect.

2.2 Part 2: characterization

Several design-flows include a step used to characterize the specification of an application. The main objective of characterization is to extract relevant information from the specification to guide the designer and/or synthesis steps towards an efficient application-architecture matching. For this purpose, metrics can be efficiently used to rapidly stress the proper architecture style for the application or for part of it (sub-tasks, functions). Some of the relevant features include the wider/deeper trade-off, namely the ratio of explicit parallelism versus the pipeline depth, the necessity of complex control structures, the requirements in terms of local memories and specific bandwidth, and the need for processing resources for specific computations or address generation.

Contrary to partial available approaches, we consider that an efficient characterization step should include all the following requirements:

- Independence & flexibility: during the characterization step the designer should have the option to specify or not an architectural target since the objective of this step is to guide either the choice of an existing architecture or the construction of a new one. Moreover, the implementation of new metric computations must be easy;
- Hierarchy & multi-granularity: enables the characterization of the different ganularity levels: e.g i) loop body, ii) loop, iii) sequence of loops, it also offers the possibility to reuse characterizations for different "mappings" of a given graph (e.g. various calls of the same sub-function);
- Data and control dependency analysis: the information about critical paths at different levels of granularity is required for in-depth parallelism characterization.

2.2.1 Review of existing metrics

Recently works dealing with metrics in the domain of high-level synthesis [16], [17] and hardware software co-design [18], [19], [20] have been proposed. In [16] the metrics provide algorithm properties regarding a hardware implementation; the quantified metrics address the concurrency of arithmetic operations based on uniformed scheduling probabilities and the regularity that measures the repetition rate of a given pattern. In [17], some probability based metrics are proposed to quantify the communication link between arithmetic operators (through memory or registers). These metrics focus on a fine grain analysis and are mainly used to guide the design of data-paths, especially to optimize local connections and resource reuse. The metrics from [18] are computed at the functional level to highlight resource, data and communication channel sharing capabilities in order to perform a pre-partitioning resulting in function clustering to guide the next design step (hardware/software partitioning). The main issue is the placement of close functions on the same component in order to optimize communications and resource sharing. An interesting method for processor selection is presented in [19]. Three metrics representing the orientation of functions in terms of control, data transformation and data accesses are computed by counting specific instructions from a processor independent code. Then a distance is calculated, with specific characteristics of processors regarding their control, bandwidth and processing capabilities. In this framework a coarse and fixed granularity level is considered and the target is limited to predefined processors. Moreover the technique does not take into account instruction dependencies and there is no detail about the different types of memory accesses regarding the abstract processor model used. However we can reuse the concept of distance during the design steps located at lower levels. Finally, in [20] finer metrics are defined to characterize the affinity between functions and three kinds of targets: GPP, DSP and ASIC. The metrics are the result of the analysis and counting of C code instructions in order to highlight instruction sequences which can be DSP-oriented (buffer circularity, MAC operations inside loops,...), ASIC-oriented (bit level instructions) or GPP-oriented (conditional or I/O instructions ratio). Then a HW/SW partitioning tool is driven by the affinity metrics. Like in [19] these metrics are dedicated to HW/SW partitioning, they do not exploit instruction dependencies and address a fixed (C procedures) granularity. Moreover, the locality of data bandwidth is not clearly taken into account.

2.2.2 Contributions

Although a number of existing works dealing with metrics can be found in the literature, some important features are not yet covered. In our work we propose to fulfill this requirements by

means of new metrics which are detailed in section 4.

Firstly, the analysis performed in other works is generally dedicated to a class of applications and architectures; our framework provides the designer with a generic library UAR (User Abstract Rules, described in 3.3.4) that can be more or less specialized to offer either independency or to match a given architecture. Secondly, regarding the heterogeneity of applications, different pieces of an application can present various features, so metrics at different levels of granularity can help to localize parts with specific features (e.g. parallelism). As explained later, our metrics are computed at each level of granularity and thus are naturally available at all levels from the leaf DFGs (Data-Flow Graph) to the complete HCDFG representing the whole C code. Finally, the system metrics from [20, 19] do not address data dependencies so can not include the critical paths during the parallelism analysis; our metric computation includes this feature.

The characterization step implemented in Design Trotter analyzes the functions of an application in order to determine the *orientation* and the *criticity* of a function. The orientation indicates if a function is processing, control or memory oriented. The criticity indicates the average parallelism available in a function. For that purpose a set of metrics has been defined, it is presented in section 4.

3 HCDFG Specification

The aim of the HCDFG model is to represent a function in a way that facilitates its estimation and exploration. The notion of function differs according to the designer point of view. In our work two specification models are considered. The first one, HFSM-HCDFG is based on Hierarchical Finite State Machines, the second one, TG-HCDFG is based on task-graphs (TG).

3.1 HFSM-HCDFG

This type of representation is presented in Fig. 2. The HFSM model has been used in the EPICURE project [21]. The hierarchical decomposition at the system-level is made with Esterel Studio [15]. The specification is based on a two level decomposition approach: event-based with Esterel and calls to C functions. During the specification step, the designer inserts calls to C functions (in the states of the system) when he considers that the HFSM model is no longer suitable. Then, regarding the model presented in Fig. 2, the specification corresponds to the "Process" view. In this view the representation is based on a graph of functions enabling sequentiality, mutual exclusion and parallelism. The designer is responsible for organizing the function setup and for defining their granularity. Finally, the "Function" view corresponds to a specific function, described with the

HCDFG model.

3.2 TG-HCDFG

The other type of specification is related to a classic real-time specification model, i.e., a task-graph. This type of specification is used to performed HW/SW partitioning and real-time scheduling considering cyclic and a-cyclic tasks [5]. This model is described in Fig. 3. It is composed of four levels: the system-level (describing the inputs/outputs constraints), the task-graph, the function graph and the HCDFG level (used to describe a function). We use the Radha Ratan model [22] to specify the task-graph.

3.3 The HCDFG model

3.3.1 HCDFG basics

As previously mentioned (see 2.1.2), the HCDFG model is used to describe the application when the designer evaluates that a control-data flow is well suited at a given hierarchy level during the specification procedure. The HCDFG generation is based on GCC from which a parser has been devised in order to extract and structure only relevant information for design exploration. Namely we build a HCDFG using the following principles :

- Single assignment with comprehensive renaming rules. Firstly we eliminate false dependencies by introducing dependence edges. Then the name of scalar or array variables is built by combining the original name with integer values that are incremented after each read and write accesses (cf. 3.4.2);
- The initial hierarchical structure is used in order to preserve data locality;
- The code hierarchy is also exploited to perform a component based-approach through graph patterns usage. A given C function/block is seen as a single graph-component (HCDFG) that can then be instantiated several times distinguished with specific numbers;
- An efficient multi-dimensional data representation;
- Data-transfer and data-processing nodes are mapped onto a generic library which can be personalized depending on the target architecture (e.g., a graph-component can be associated to a processing unit);
- To summarize, all the elements of the model are represented by graphs. Thus, during the

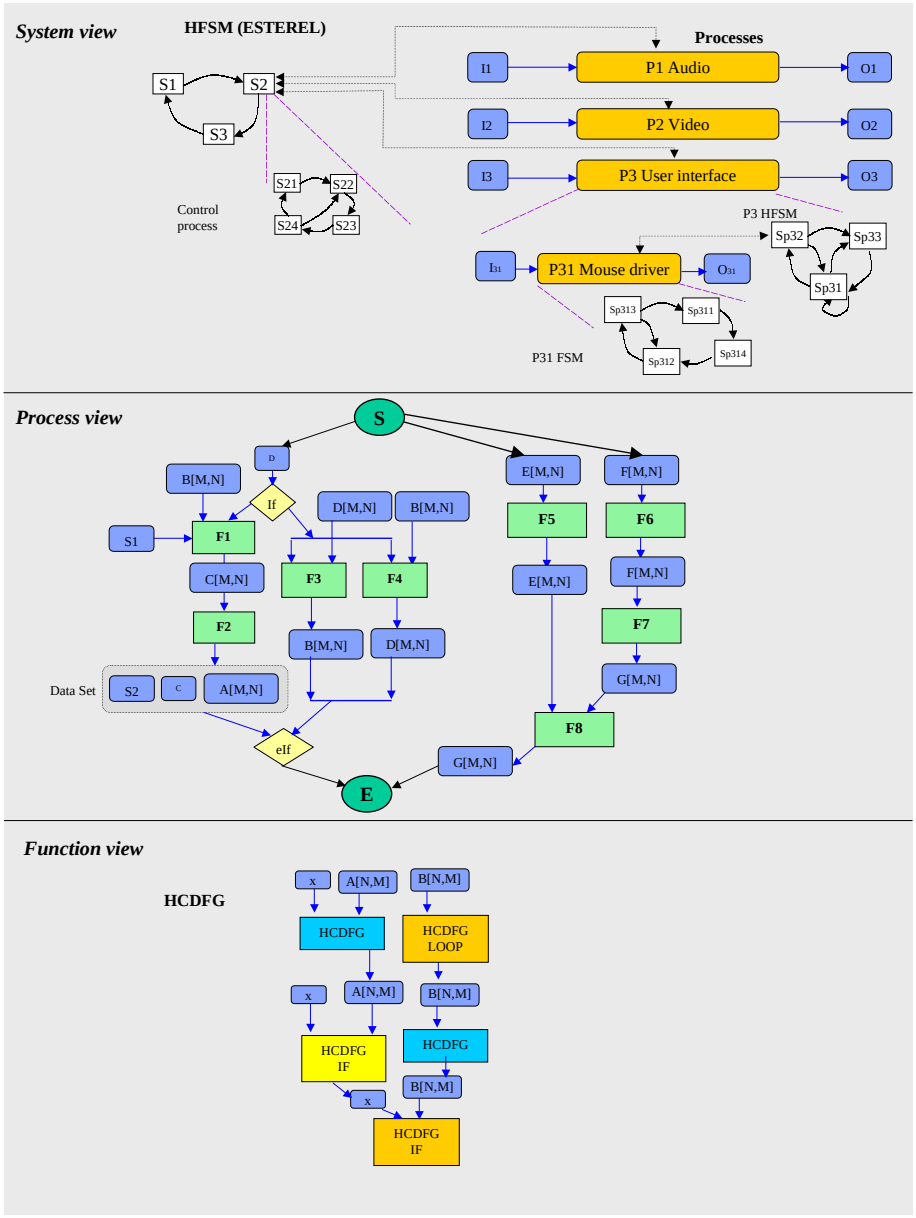

Figure 2: *Specification example with the HFSM-HCDFG model.*

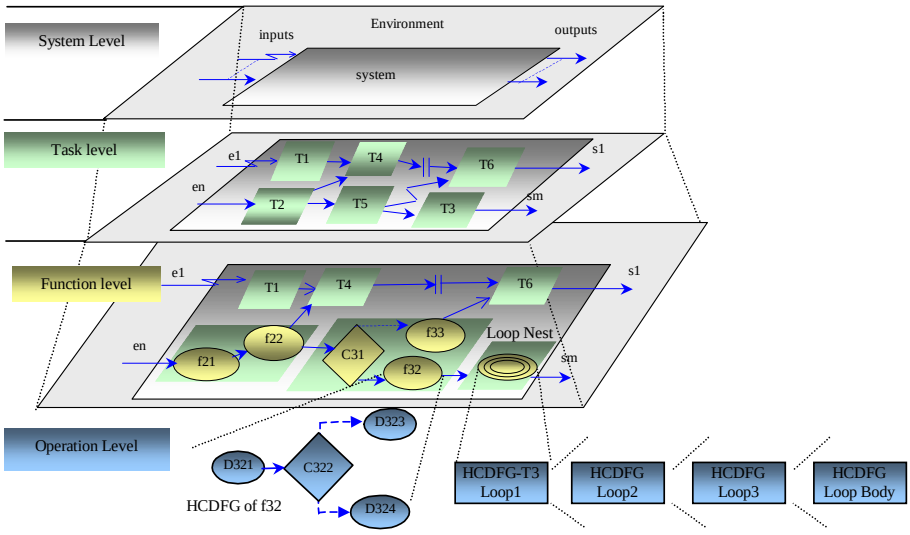

Figure 3: *Specification example with the TG-HCDFG model.*

characterization and estimation steps, important elements and structures can be easily identified by means of our uniform model.

3.3.2 Definitions

The HCDFG model enables the representation of a function in the form of a hierarchical graph containing control structures and processing operations manipulating scalar and array data. This graph has been defined in order to make the characterization and estimation steps efficient. The required information and the results are stored as attributes in the graph. Attribute examples are the ASAP and ALAP dates of nodes, hierarchy levels for memory nodes, metric results and so on. Each function described in the C language is parsed to a HCDFG, an example is given in Fig. 4. A HCDFG is composed of elementary nodes (processing, memory, control), dependence edges (control, data) and graphs that can be hierarchical. The different key features are detailed hereafter.

Elementary nodes A *processing node* (processing vertex) can represent several types of operations: fine grain arithmetic /logic operations but also coarse grain computations (MAC, Butterfly, FIR, DCT, Pixel Shader,...). As the association between operations and resources is defined in the UAR file (described in 3.3.4), it is possible to reference the name of a sub-graph (instance of a graph pattern) in the UAR, indicating that a resource is dedicated to this computation. In that case the graph is seen as a black box, which may already have been estimated (in particular the cycle budget). Processing nodes can be seen on the right part of Fig. 4.

A *memory node* (memory vertex) represents a data-transfer. The main node parameters are the transfer direction (read / write), the data format and the hierarchy level which can be fixed by the designer. Data are explicitly represented by nodes in the graph, and are not, like in many other models, associated to edges. The advantage of such a model is to not duplicate information concerning data and to not overload the specification. For multi-dimensional data (arrays, vectors), addressing mechanisms are explicitly represented in the graph through index DFGs. Memory vertices (scalar and array) are exposed on the right and left part of Fig. 4 respectively.

A *conditional node* represents a test operation (if, case, loops, etc.).

Dependence edges Three types of oriented edges are used to indicate scheduling constraints.

A *Control dependency* indicates an order between operations without data-transfer, for instance a test operation must be scheduled before mutual exclusive branches.

A *Scalar data dependency* between two nodes A and B indicates that node A uses a scalar data produced by node B.

A *Multidimensional data dependency* is a data-dependency where the data produced is not a scalar but an array. For instance such an edge is created between a *loop* CDFG (Control-Data Flow Graph) reading an array transformed by another *loop* CDFG.

Graphs There are six kinds of graphs.

A *DFG* is a graph which contains only elementary memory and processing nodes. Namely it represents a sequence of non-conditional instructions of the 'C' code. The graph on the right part of Fig. 4 is a DFG.

A *CDFG* is a graph which represents a test or a loop pattern with associated DFGs. A CDFG is made of: two control nodes (begin and end) which indicate the type of the structure (*if, switch-case, while, do-while and for*), an evaluation graph, plus an evolution graph in the case of a FOR structure, and finally one or more H/CDFGs which represent the processing conditioned by the control node. Moreover, an attribute enables the saving of the execution probability of each branch (for control structures). The probabilities can be obtained by profiling or can be specified directly by the designer through an interactive and user-friendly interface. The graph in the center of Fig. 4 is a CDFG.

An *Evaluation graph* produces a boolean data, it corresponds to the computation of a condition. The boolean data node is connected to the control node by an order edge.

An *Evolution graph* is found in FOR structures. It represents the increment mechanism of loop indexes (only affine increment mechanism are allowed). An evolution graph corresponds to

the computation of the index increment. The data node representing the index of a FOR loop is connected to the control node by an order edge.

An *Index graph* represents index computations (e.g., $i + (j * 2)$ in array[i+(j*2)] = 0). This is a key feature for detecting the need for address generation units (AGUs).

A *HCDFG* is a graph which contains elementary conditional nodes, HCDFGs and CDFGs. It represents the application hierarchy, i.e., the nesting of control structures and graphs executed in sequential or parallel patterns. The graph on the left part of Fig. 4 is a HCDFG.

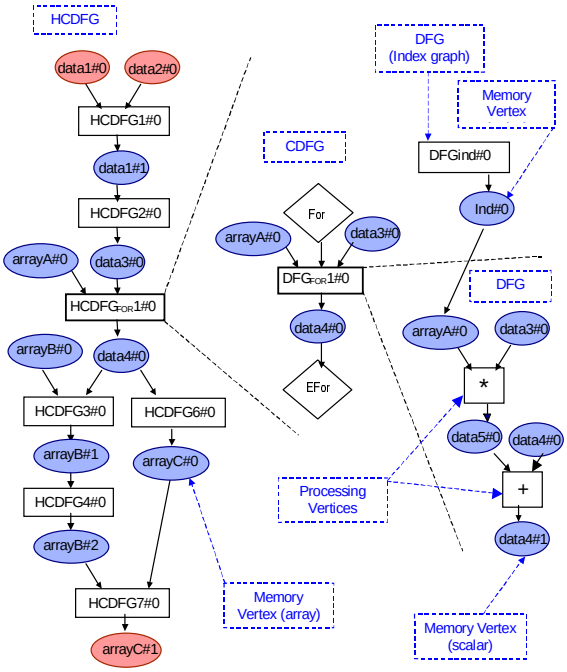

Figure 4: *Elements of a HCDFG.*

3.3.3 Graph creation rules

The composition principle is quite natural. The graph is traveled with a depth-first search algorithm. When no more conditional nodes are found, a DFG is built. Then a H/CDFG is created each time a conditional node is found in the upper hierarchy level. Another stop condition is encountered when the name a graph pattern can be associated to a function already referenced in the architectural model (UAR). In order to facilitate the estimation process, classic CDFG patterns have been defined to identify rapidly the usual nodes like *loop, if, etc.*.

3.3.4 Architecture definition

The designer defines a set of rules, named "UAR" (User Abstract Rules) which aims at describing an architectural model for both characterization (optional) and exploration (at system-level or for

HW and SW projections). When no architecture has been selected the designer can describe any kind of abstract architecture to start the exploration process. Then the UAR can be refined using some feedback from the exploration process. As the UAR model is flexible, the designer can also describe an existing architecture.

The processing part of the architecture is characterized by the type of available resources : ALU, MAC, etc. and the operations they can perform; a number of cycles is associated to every type of operator. The granularity of the operators is not fixed, and coarse grain operators (MAC, FFT or more complex functionality) can be described and associated to operations/functions of the HCDFG description of the application. Regarding the memory part, the user defines the number of levels (L_i) of the hierarchy and the number of cycles (l_i) associated for each type of access.

Fig. 5 shows an example of two User Abstract Rules files. The first one (left) is the initial file where all resources have a latency equal to one cycle. When the designer starts to refine the architectural model (using results given by the system-level estimation), he can add new types of resources or specify resource latencies like in the second file (right). Thus the designer may improve his analysis by means of system-level estimation.

```
<LIBRARY> system
  <OPERATOR> Alu
      <OPERATIONS>
          "+","-","*","/",
          "<=","=","!=",">="
      <ENDOPERATIONS>
      <ATTRIBUTES>
          latency:cycle:=1
          datawidth:INT:=8
      <ENDATTRIBUTES>
  <ENDOPERATOR>
  <OPERATOR> Mac
      <OPERATIONS> "*+"
      <ENDOPERATIONS>
      <ATTRIBUTES>
          latency:cycle:=1
          datawidth:INT:=8
      <ENDATTRIBUTES>
  <ENDOPERATOR>

/* MEMORY */
  <MEMORY> RAM_DP
      <ATTRIBUTES>
          access_mode:=rw
          latency_read:cycle:=1
          latency_write:cycle:=1
      <ENDATTRIBUTES>
  <ENDMEMORY>
<ENDLIBRARY>
```

```
<LIBRARY> system
  <OPERATOR> Alu
      <OPERATIONS>
          "+","-","*","/",
          "<=","=","!=",">="
      <ENDOPERATIONS>
      <ATTRIBUTES>
          latency:cycle:=2
          datawidth:INT:=32
      <ENDATTRIBUTES>
  <ENDOPERATOR>
  <OPERATOR> Mac
      <OPERATIONS> "*+"
      <ENDOPERATIONS>
      <ATTRIBUTES>
          latency:cycle:=3
          datawidth:INT:=32
      <ENDATTRIBUTES>
  <ENDOPERATOR>

/* MEMORY */
  <MEMORY> RAM_DP_LEVEL_1
      <ATTRIBUTES>
          access_mode:=rw
          latency_read:cycle:=1
          latency_write:cycle:=2
      <ENDATTRIBUTES>
  <ENDMEMORY>
  <MEMORY> RAM_DP_LEVEL_2
      <ATTRIBUTES>
          access_mode:=rw
          latency_read:cycle:=2
          latency_write:cycle:=3
      <ENDATTRIBUTES>
  <ENDMEMORY>
<ENDLIBRARY>
```

Figure 5: UAR file examples (left: first approach; right: refinement)

3.4 Java data structure

3.4.1 Two step construction

This section briefly presents how the graph-component approach is implemented in Java. Basically, the internal model is built in two main steps as illustrated in Figs. 6, 7 and 8. The first step consists in translating the C code into the HCDFG grammar and the second one converts the HCDFG description into a Java data-structure. The main advantage of this construction is the independence during the first step from the architectural model defined in the UAR file. The first step extracts the relevant information and the second one exploits it regarding the target architecture.

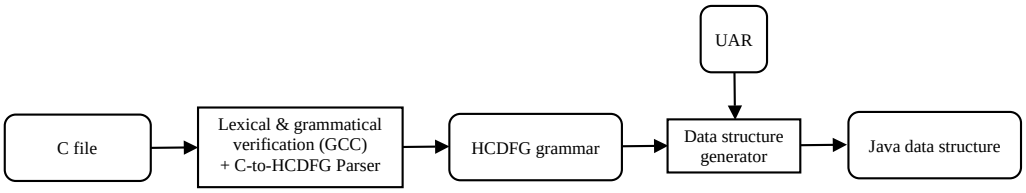

Figure 6: *From C to data structure*

3.4.2 Data and graph representation

The main issues of a graph representation for system design is the size of the resulting structure and its capabilities for information retrieving and synthesis. To cope with this trade-off, we first implement a graph-component approach where a single graph definition can be mapped several times. The second point is the data representation: in order to limit the memory requirements for representing all data and accesses to these data, we use a single assignment rule. It implies that a data is represented only once and that each data access (R/W) can be clearly identified through false data-dependencies elimination. This is done by means of a specific formalism: data names are kept but are extended with two suffixes. For instance, if we consider accesses to data A, the node A♯k%n is interpreted as follows : the "♯ k" symbol indicates the k^{th} write access to data A and the "% n" symbol indicates the n^{th} read access to data A following the k^{th} write acces.

For example, the following C code:

```
y = a*b;
b = y+2;
c = y;
b = c + 1;
```

will lead to the following *accessEdge* elements:

```
a#0%0 , b#0%0 , y#1 ; y#1%0 , b#1 ; y#1%1 , c#1 ; b#2 , c#1%0
```

To avoid size explosion, the array accesses are treated in the same way, independently from the index values. Firstly the index computations are considered during the characterization step as processing operations since they also can impact the critical path computation. Secondly, they are taken into account during the scheduling/combinations/design space exploration step (Fig. 1) since they introduce data dependencies. Finally, the index evolution is considered during the memory size fine-estimation based on polyhedral computation [23]. This is not the topic of this paper but this point also explains our choices. Briefly, the default memory size estimation considers array definitions, the dynamic memory size estimation has been integrated in our tool as an option that takes benefit from our graph structure and scheduling algorithms to implement a polyhedral-based memory size estimation method.

Examples of transformations from a C file to a HCDFG and from a HCDFG to the data structure are given in Fig. 7 and Fig. 8 respectively.

3.4.3 Interactive facilities

The aim of our work is to provide the designer with a fast and easy to use analysis tool. Thus the designer can modifiy, once HCDFGs have been built, the following parameters :

- *Constant values* such as loop bounds, array size, etc. The designer can also choose a set of values in order to observe the evolution of the metrics regarding a given constant;
- *Test probabilities* such as IF or CASE conditions which are equiprobable by default. When such a modification occurs the occurrence probabilities of all graphs are updated and stored.

The result data are stored in an XML file to facilitate exchanges with other tools, to keep the graph hierachy and to get a generic graphic interface.

4 Characterization

The functions composing an application can have very different features in terms of orientation (processing, control, memory) and potential parallelism. Characterizing functions has two objectives: i) guide the designer in his architectural choices and ii) guide the function estimation step in order to use the most adequate scheduling algorithms [1].

We have defined the following metrics:

Figure 7: *From C to HCDFG.*

Figure 8: *From HCDFG to data structure.*

- orientation metrics: *Memory Orientation Metric (MOM)*, *Control Orientation Metric (COM)*.
- criticity metric: γ.
- data reuse metric (DRM) and Hierarchical Data Reuse Metric (HDRM).

4.1 Definitions and memory model

Before to give the metric formula details, we introduce some terms which will be used in what follows.

- **Processing**: includes computation operations (ALU, MAC...), addresses computation (which can be performed on ALUs or specific units such as AGUs *Address Generator Unit*) [24], deterministic control (e.g. constant loop bounds that can be eliminated by unfolding);
- **Control**: includes tests, namely control operations that are not computable by a compiler (data-dependant control);
- **Memory**: includes read/write data-transfers. We distinguish the size, read/write access timing and the simultaneous number of accesses. Moreover the memory can be hierarchical. In this work, we have considered two levels from an architectural point of view: i) the local memory which includes cache units and internal registers and ii) the global memory which include RAM, ROM, hard-drives, cf. section 4.1.1.

4.1.1 Memory model

A key point in the following sections is the notion of local and global accesses from a graph point of view. This notion is used during the characterization and the estimation steps. We have distinguished several types of memory nodes (the type is saved as an attribute for each node in a graph):

- N1: inputs/outputs: data identified as inputs/outputs of a graph;
- N2: temporary data: produced by internal processings;
- N3: reused data: input data (N1 subset) reused in a graph (detected with suffixes values);
- N4: accumulation data: annotated with pragmas during the specification.

At the system-level the local memory size associated to the processing unit is not yet known. It is therefore necessary to define a general notion of locality related to the application and not to the architecture. A global access is a data transfer to/from a data which is defined outside the

considered graph. A local access is a data transfer to/from a data defined in the considered graph (register-register transfer from an architectural point of view). N1 data are always global, N4 data always local, N2 and N3 data are initially local but can generate local/global swapping during the scheduling steps if the local memory size is limited [1].

4.2 Computation of the metrics for a function (i.e., for a HCDFG)

The computation of the metrics is performed hierarchically, with an ascending approach. First, the metrics are computed for the leaf graphs (DFGs), then for control graph (CDFG) and hierarchical graphs (HCDFGs) using mutual-exclusive rules (CDFGs) and parallel and sequential rules (CDFGs, HCDFGs). The metrics at a level i of the hierarchy are not simply obtained by a combination of metrics at level i-1, instead they are computed using the features stored a level i-1 such as the number of processing operations, memory accesses, control nodes and the critical paths as explained in sections 4.3.1, 4.3.2, 4.4.3 and 4.4.3.

Note also that on the one hand the orientation metrics (section 4.3) are computed without knowledge of the node ASAP/ALAP dates, only by counting the relevant elements. On the other hand the criticity (section 4.4) and (H)DRM (section 4.5) metric computations use ASAP/ALAP dates computed with UAR features.

Abbreviation	Meaning
Np	number of processing operations
Nc	number of tests (control operations which can not be eliminated at compile time)
Nm	number of global memory accesses operations with Nb_p = number of operations of type ALU, Macs,...+ Index computation + tests
tr/fa	graphs of mutually exclusive branches in "IF-THEN-ELSE" structures
p_{tr}, p_{fa}	execution probabilities of true and false branches respectively (obtained by profiling or specified by the designer, equals 0.5 by default)
for	graph of the core of a "FOR" structure
eval	graph of the evaluation part of a "FOR" structure
evol	graph of the evolution part of a "FOR" structure
N_{ite}	number of iterations in a loop structure

Table 1: *Abbreviation used*

4.3 Orientation metrics

For the three orientation metrics we firstly give a general formula. Then we detail the computations for DFGs, CDFGs (IF-THEN-ELSE and FOR) and HCDFGs. The computation of SWITCH, WHILE and DO-WHILE structures are not presented since they are generalization of IF-THEN-ELSE and FOR computations. The abbreviations used are presented in table 1.

4.3.1 Memory Orientation Metric (MOM)

MOM indicates the frequency of memory accesses in a graph. The general formula for MOM is the ratio between the number of global memory accesses and the number of global memory accesses plus the number of processing operations. Its value is bounded in the interval [0;1]. High MOM values indicates that processing operations are applied to new data (i.e., data entering the graph, as opposed to data computed previously which might reside in the local memory). The more MOM $\rightarrow 1$, the more the function is data transfer oriented (if MOM $= \frac{3}{4}$ then a processing operation requires, in average, three memory accesses). In the case of hard time constraints, high performance memories are required (large bandwidth, dual-port memory,...) as well as an efficient use of memory hierarchy and data locality [25].

- **DFG MOM**

  For a DFG MOM is computed as follows:

$$MOM = \frac{Nm}{Nm + Np}$$

- **IF-THEN-ELSE MOM (and SWITCH by extension)**

  For this structure MOM is given as the ratio between the number of memory operations in the branches multiplied by their respective execution probabilities and the sum of all the operations in the branches multiplied by their respective execution probabilities.

$$MOM_{IF} = \frac{Nm_{tr} * p_{tr} + Nm_{fa} * p_{fa} + Nm_{eval}}{\sum_{x=p,c,m}(Nx_{tr} * p_{tr} + Nx_{fa} * p_{fa} + Nx_{eval})}$$

- **FOR MOM (and WHILE DO-WHILE by extension)**

  For this structure MOM is given as the ratio between the number of memory operations in each part of the loop (evaluation, core and evolution) and the sum of all the operations in each part.

$$MOM_{FOR} = \frac{Nm_{eval} + Nmfor + Nm_{evol}}{\sum_{x=p,c,m}(Nx_{eval} + Nx_{for} + Nx_{evol})}$$

- **HCDFG MOM**

  For a HCDFG MOM is given as the ratio between the sum of all memory operations in the

sub-graphs of the HCDFG and the sum of all the operations in the sub-graphs.

$$MOM_{HCDFG} = \frac{\sum_{\text{sub-graphs j}} Nm_j}{\sum_{\text{sub-graphs j, x=p,c,m}} Nx_j}$$

4.3.2 Control Orientation Metric (COM)

COM indicates the appearance frequency of control operations (i.e., tests that cannot be eliminated during compilation) in CDFGs and HCDFGs (since there is no test within a DFG).

The general formula of COM is the ratio between the number of tests and the total number of operations including processing operations, tests and accesses to global memories. COM values are bounded in the interval [0;1]. The closer to 1 COM is, the more the function is control dominated, so needs complex control structures (if COM = $\frac{3}{4}$ then 1 operation out of 4 is a non-deterministic test). It also indicates that the use of the pipeline technique is not efficient for such functions.

- **DFG COM**

  For a DFG, COM equals 0 since there is no control in a DFG.

- For a CDFG the generic formula for COM is computed as follows:

$$COM = \frac{Nc}{Np + Nc + Nm}$$

- **IF-THEN-ELSE COM (and SWITCH by extension)**

  For this structure COM is given as the ratio between the number of control operations in the branches multiplied by their respective execution probabilities and the sum of all the operations in the branches multiplied by their respective execution probabilities.

$$COM_{IF} = \frac{Nc_{tr}p_{tr} + Nc_{fa}p_{fa} + Nc_{eval}}{\sum_{x=p,c,m}(Nx_{tr} * p_{tr} + Nx_{fa} * p_{fa} + Nx_{eval})}$$

- **FOR COM (and WHILE DO-WHILE by extension)**

  For this structure COM is given as the ratio between the number of control operations in each part of the loop (evaluation, core and evolution) and the sum of all the operations in each part.

$$COM_{FOR} = \frac{Nc_{eval} + Ncfor + Nc_{evol}}{\sum_{x=p,c,m}(Nx_{eval} + Nx_{for} + Nx_{evol})}$$

- **HCDFG COM**

  For a HCDFG COM is given as the ratio between the sum of all control operations in the sub-graphs of the HCDFG and the sum of all the operations in the sub-graphs.

$$COM_{HCDFG} = \frac{\sum_{\text{sub-graphs j}} Nc_j}{\sum_{\text{sub-graphs j, x=p,c,m}} Nx_j}$$

4.4 Criticity metric

The approach used in our methodology consists in estimating the most critical functions first (the less critical ones may reuse the resources allocated to the most critical ones and are therefore estimated after). Criticity is defined by the γ metric such as:

$$\gamma = \frac{\text{Nb processing and memory accesses operations}}{\text{Critical Path}}$$

The critical path of a DFG is defined as the longest chain of sequential operations (expressed in cycle number). The critical path for a function is computed hierarchically by combining the critical path of its HCDFGs.

γ indicates the average parallelism available at a specific hierarchy level: lets consider a HCDFG composed of 5 identical parallel DFGs. If each DFG is internally executed in a sequential manner then γ for each DFGs equals 1 but γ for the whole HCDFG equals 5.

A function with a high γ value can benefit from an architecture offering high parallelism capabilities. On the other hand, a function with a low γ value has a rather sequential execution. In that case the acceleration of this function can be made via temporal parallelism (e.g., long pipeline), depending on COM value. From a consumption point of view a function with a high parallelism offers the opportunity to reduce the clock frequency by exploiting the spacial parallelism.

4.4.1 IF-THEN-ELSE γ (and SWITCH by extension)

The criticity metric for this structure is the ratio between the number of operations in the sub-parts of the control structure multiplied by their respective probabilities and the sum of the critical paths of the sub-parts.

$$\gamma_{IF} = \frac{\sum_{x=p,c,m}(Nx_{tr} * p_{tr} + Nx_{fa} * p_{fa} + Nx_{eval})}{p_{tr} * CP_{tr} + p_{fa} * CP_{fa} + CP_{eval}}$$

4.4.2 FOR γ (and WHILE DO-WHILE by extension)

The criticity metric for this structure is the ratio between the sum of the operations in each part of the loop and the sum of the critical paths of the sub-parts.

$$\gamma_{FOR} = \frac{\sum_{x=p,c,m}(Nx_{evol} + Nx_{for} + Nx_{eval})}{CP_{evol} + CP_{for} + CP_{eval}}$$

4.4.3 HCDFG γ :

For a HCDFG representing a serial execution of sub-graphs the criticity metric is given as the ratio between the sum of all control operations in the sub-graphs of the HCDFG and the sum of all the critical paths.

$$\gamma_{\text{serial}} = \frac{\sum_{\text{sub-graphs j, x=p,c,m}} Nx_j}{\sum_{\text{sub-graphs j}} CP_j}$$

For a HCDFG representing a serial execution of sub-graphs the criticity metric is given as the ratio between the sum of all control operations in the sub-graphs of the HCDFG and the longest of all the critical paths.

$$\gamma_{\text{parallel}} = \frac{\sum_{\text{sub-graphs j, x=p,c,m}} Nx_j}{MAX_{\text{sub-graphs j}}(CP_j)}$$

4.5 DRM and HDRM metrics

4.5.1 DRM metric

Balancing processing and memory access operations is a critical point in system design to face the memory bandwidth bottleneck (as compared to the processors performances). The balance can be obtained through the use of scheduling algorithms adapted to the function orientation. In order to guide function analysis and DFG scheduling we have defined a metric called *DRM: Data Reuse Metric*. At system-level the only memory hierarchy information available is naturally extracted from the algorithmic memory hierarchy (i.e., from the high-level language specification). This metric takes into account the local memory size, which has to be fixed by the designer or estimated. The estimation is performed as explained in the next paragraph. Moreover, as we want to obtain estimates rapidly, methods such as clique coloring have been discarded. The DRM metric gives the global/local accesses ratio. A local access which produces a memory conflict (local memory full) involves a global read and write, thus the number of extra global accesses is $\beta \times EGT$ with β the average number of cycles required by an access to the global memory (main memory

with or without cache capabilities.) In the case of a single main memory which requires only one cycle per access $\beta = 2$ (read plus write).

We use average data lifetime to estimate the quantity of data alive in each cycle, from which the minimum memory size can be derived. Minimum and maximum data-life of data d are defined as follows:

$$MinDL(d) = ASAP(d^n) - ALAP(d^1) + 1$$

$$MaxDL(d) = ALAP(d^n) - ASAP(d^1) + 1$$

where $ASAP$ and $ALAP$ are the earliest and latest scheduling dates respectively, d^1 and d^n the earliest and the latest read access to data d for a given time constraint respectively. The average data-life of data d is then given by:

$$AvDL(d) = \frac{1}{2}\left(MinDL(d) + MaxDL(d)\right)$$

Finally, the number of data alive per cycle is given by:

$$AAD = \frac{1}{T}\sum_{d} AvDL(d)$$

where T is the number of cycles allocated to the estimated function (the estimation process is based on a time constraint scheduler, using several time constraints [1]). The number of local transfers turning into global transfers because of a too small local memory is given by:

$$EGT = \begin{cases} (AAD - UM)T \; if \; AAD > UM \\ 0 \; otherwise \end{cases}$$

where UM is the local memory sized. UM can be defined by the user, its default value is AAD.

If we consider a memory hierarchy with the following characteristics, Level 1 (L1): $latency1(l1) = 1cycle$, L2: $l2 = 2cycles$, L3: $l3 = 3cycles$, and if MR_i is the miss ratio of the cache level i, then:

$$\beta = 1.(1 - MR_1) + 2.MR_1.(1 - MR_2) + 3.MR_1.MR_2$$

If we generalize to K levels of hierarchy (including hard disk and network for instance) we obtain:

$$\beta = \sum_{k=1}^{K} \left(l_k.(1 - MR_k). \prod_{j=1}^{K-1} MR_j \right)$$

Finally the DRM metric is given by:

$$DRM = \frac{N1 + \beta.EGT}{N1 + N2 + N3 + N4}$$

The DRM metric is computed for DFGs and can be used for selecting the most appropriate scheduling algorithm during the estimation step [26].

4.5.2 HDRM metric

The hierarchical DRM (HDRM) extends the DRM metric to inter-HCDFG data reuse (remember that a HCDFG is a graph which contains elementary conditional nodes and parallel and serial HCDFGs and CDFGs). Its computation is general and used for all hierarchical estimations, it is based on two by two HCDFGs clustering. The principle is given in Fig. 9: A1 and A2 are the amount of exclusive input data read by the HCDFG1 and HCDFG2 respectively. A12 quantifies the input data which are common to both graphs and A3 is the amount of result data from the first graph transmitted to the second one. Thus the HDRM computed by the formula 8 provides the ratio of reused data between two HCDFG which can be parallel (A3 = 0), sequential (A12=0) or a combination.

The HDRM metric is computed as follows:

$$HDRM = \frac{A_3 + A_{12}}{A_1 + A_2 + A_3 + A_{12}}$$

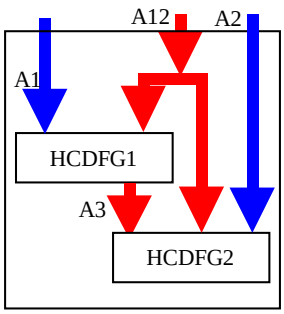

Figure 9: *Illustration of the HDRM metric.*

HDRM = 1 means that the reuse ratio is maximal: all data read by the HCDFG2 are shared or produced by HCDFG 1, it also denotes that a local memory could be efficiently implemented. On the other hand DRM12 = 0 means that no data-reuse is available for memory optimization. In multimedia applications, data reusing has a important impact especially because the optimization opportunities are mainly due to memory management of loop nests. According to that point, we

observe that the HRDM computation can be computed while considering HCDFG1 and 2 as two successive loop iterations.

Illustrative example

We now present how the HDRM computation can be applied to the six nested loops of a classical motion estimation algorithm. Our aim here is to use the HDRM in order to highlight data reuse between subsequent iterations at a given level of hierarchy. If we refer to the illustration in Fig. 9, it means that we virtually consider a loop unrolling where each iteration is embodied by a HCDFG. The HDRM has been computed for the different loop levels: column and row of a $n*n$ (with n=8), column and row of the $2m+n-1*2m+n-1$ reference window (with m=16) and column and row of a WxH frame (QCIF format: W=174, H=144). Table 2 presents the results regarding the Ai and HDRM values for each hierarchical level. (OF and NF mean Old and New frame respectively). The last column shows the size of the memory candidate to store the old frame data at each level of hierarchy. We observe that the best data reuse opportunities are available when the HCDFG-core of the following loop indices are considered: column of the reference window (HDRM = 88%), row of the reference window (HDRM = 81%) and row of the frame (HDRM=87%). It means that including an ad hoc memory hierarchy to locally store these highly reused data can provide high performance and power optimizations.

Level	A1-OF	A1-NF	A2-OF	A2-NF	A12-OF	A12-NF	A3	HDRM	OF Memory size
block column	1	1	1	1	0	0	1	0,20	1
block row	8	8	8	8	0	0	1	0.03	n
window column	8	0	8	0	56	64	0	0,88	$n*n$
window row	39	0	39	0	273	64	0	0,81	$(2m+n-1)*n$
frame column	312	312	312	312	1209	1209	0	0,66	$(2m+n-1)^2$
frame row	1408	1408	1408	1408	5456	25336	0	0,87	$W*(2m+n-1)$

Table 2: *HDRM metric Motion estimation theoretical results. OF: Old Frame, NF: New Frame*

Final remark on the metrics

Compiler optimizations and transformations can have strong impacts on the final code and it would be therefore desirable to evaluate these impacts as regard to the metrics. This point is out of scope of this paper, however it should be possible to evaluate these impacts by accessing post target-independent optimized code and to characterize this code. Finally by comparing the two characterizations, the compiler impact could be evaluated.

5 Results

We have applied the previously defined metrics to functions widely used in embedded systems. Hereafter we present results computed with the aim to be as much as possible independent from any architecture, namely we consider an algorithmic characterization based on a unspecialized UAR. The following examples are detailed: a wavelet transform (DWT) and a 2D-DCT transform, a G722 audio decoder, a TCP protocol and two video applications matching pursuit (MP): Object Motion Detection (OMD). Regarding the OMD application we also provide estimations results on a FPGA target.

5.1 DWT

The DWT algorithm [27] has been implemented using the lifting scheme. It is composed of two sequential blocks (C sub-functions translated into HCDFGs, level N-1) which operate sequentially in the horizontal and vertical dimensions respectively. Each block is made of 6 sub-blocks (C sub-functions translated into HCDFGs, level N-2). The C code is made of one function (translated into a HCDFG, level N) englobing the code for the sub-blocks. The lower level of granularity depend on the structure of each sub-blocks whose metrics are automatically computed like for upper levels. The delay for the whole characterization step equals 500ms on a PIII@700Mhz with Java implementation. Table 3 provides the results for the six functional sub-blocks and for the whole function (top graph). The first observation is that the COM metric equals zero for all graphs, since this application is composed of deterministic loops and does not contain any test. Secondly we observe that MOM values for the wavelet functional blocks are higher than 0,7; this means that more than 7/10 of operations are data accesses, so the application is clearly, at all levels, memory oriented. Finally, γ values are around 1,5 for all the functional blocks. The horizontal and vertical blocks have gamma values equal to 2,704. As the two blocks are executed sequentially, the gamma value for the whole function (top graph) also equals 2,704. This indicates that spatial parallelism is rather weak considering the fine grain sub-blocks and a that a coarse grain parallelism is available, i.e., parallelism between the horizontal and vertical blocks. By analyzing the metrics values, the designer can notice that there i) is no need for complex control structures, ii) are important needs for high data-access requirements and iii) is a coarse grain parallelism. This means that optimizations can be obtained with a pipelined architecture with possible coarse grain dedicated hardware modules providing a large bandwidth. So if high performances are required, a (programmable) dedicated hardware can be introduced within the SOC.

Sub-blocks N-2	**MOM N-2**	**COM N-2**	γ **N-2**
HFirstLiftingStepFOR11	0,721	0	1,576
HFirstDualLiftingStepFOR21	0,721	0	1,576
HSecondLiftingStepFOR31	0,721	0	1,576
HSecondDualLiftingStepFOR41	0,722	0	1,579
HScalingFOR51	0,802	0	2,136
HRearrangeFOR61	0,904	0	1,843
VFirstLiftingStepFOR71	0,721	0	1,576
VFirstDualLiftingStepFOR81	0,721	0	1,576
VSecondLiftingStepFOR91	0,721	0	1,576
VSecondDualLiftingStepFOR101	0,722	0	1,579
VScalingFOR111	0,802	0	2,136
VRearrangeFOR121	0,904	0	1,843
Blocks N-1	**MOM N-1**	**COM N-1**	γ **N-1**
Hlines	?	?	2,704
Vlines	?	?	2,704
Top-graph N	**MOM N**	**COM N**	γ **N**
DWT (top graph)	0,765	0	2,704

Table 3: *DWT characterization*

5.2 G722 Decoder

The UIT-T G722 recommendation is one of the audio part of the H320 standard for video-conference. We have studied the coder part of the application, and more specifically the adaptive predictor block (predicSup). This block is made of 8 sub-blocks (filters) which execute concurrently. The characterization results are found in table 4. We can notice that the results are quite similar to those of the previous example. First of all the COM values are very small, which indicates that there is almost no tests. Next we observe high MOM values which reflect a large number of global memory accesses. Finally the parallelism is weak at fine grain levels (between 1.33 and 2.33 for the eight sub-blocks) and increases at the highest levels of the hierarchy, since the sub-blocks execute concurrently (3.60 and 3.62 for predic and predicSup respectively). The parallelism evolution is quite similar to the DWT example since the parallelism is increasing from level N-2 to N-1 and remains stable at level N, however larger gains are obtained. By considering a cross analysis of MOM and γ we observe that in order to exploit the available parallelism ($\gamma = 3.62$) the architecture should provide enough simultaneous memory accesses since more than 70% of operations are data-transfers. By referring to the metrics the designer should select an architecture with good I/O capabilities and enough computational power to execute the sub-blocks concurrently. For example a large DSP such as the Texas Instrument TMS320C6201 coupled with a I/O co-processor featuring large FIFOs could be used.

Sub-blocks N-2	**MOM N-2**	**COM N-2**	**γ N-2**
parrecEecons	0,714	0	2,333
upzero	0,758	0,039	1,686
uppol2	0,674	0,087	2,045
uppol1	0,743	0,086	2,188
recons	0,603	0,081	2,128
filtez	0,688	0	1,375
filtep	0,5	0	2,000
predic	0,75	0	1,333
Sub-blocks N-1	**MOM N-1**	**COM N-1**	**γ N-1**
predicSup	0,738	0,037	3,602
Sub-blocks N	**MOM N**	**COM N**	**γ N**
predictorSup (top graph)	0,739	0,037	3,621

Table 4: *G722 characterization*

5.3 2D DCT

This application is a well known 2D-DCT for 8x8 image blocks, this example is interesting to illustrate metric interpretations. From a structural point of view, it is composed of two identical and sequential 1D-DCT sub-blocks (operating on lines and columns), so the corresponding graphs have the same metric values as seen in table 5. We can notice that the γ metric extracts the high degree of parallelism (5,71) provided at the lowest level of granularity (N-1). The parallelism does not increase at the second level of granularity (N) because of strict data-dependencies between sub-functions. We also observe that MOM metric is near to 0,5 compared to the DWT example where MOM is greater than 0,7. It means that the reuse of temporary local data is here much more important, it is also related to the larger degree of available parallelism.

Sub-blocks N-1	**MOM N-1**	**COM N-1**	**γ N-1**
DCT8L	0,575	0	5,714
DCT8C	0,575	0	5,714
Sub-blocks N	**MOM N**	**COM N**	**γ N**
DCT8x8 (top graph)	0,575	0	5,714

Table 5: *2D DCT characterization*

5.4 TCP

We have computed the TCP protocol metrics in order to test another kind of classical application. Each function represents a TCP state within a FSM specification. Table 6 shows the analysis result for some representative functions of TCP. We can notice that the functions have relatively high COM values denoting heavily conditioned data-flows. The MOM metric values (greater than 1/3) also indicate an important data accesses frequency. It means that these functions are control-

oriented and require high memory bandwidth. So, a suitable target architecture is a GPP powered by efficient I/O devices. There is no need for a DSP and for a complex data path structure, since the parallelism cannot be exploited since the functional blocks are clearly control oriented. Note that another very efficient architecture could be implemented using a dedicated FSM associated with fast FIFOs.

Functional blocks	**MOM**	**COM**
TCPTIMEWAIT	0,482	0,06
TCPFINWAIT2	0,534	0,055
TCPABORT	0,457	0,343
TCPwakeup	0,333	0,556
Tfinsert	0,5	0,01
TCPdodat	0,375	0,06
TCPSENT	0,508	0,320
TCPRESET	0,667	0,148

Table 6: *TCP characterization*

5.5 Matching pursuit

We have been provided with the Matching pursuit application in the context of a collaboration project with EPFL [28]. The matching pursuit application is a new compression mode which does not operate on pixels but on "atoms" representing basic patterns in a picture. This example is interesting because it is still under development: the specification is still evolving, it is therefore interesting for the designer to be able to evaluate rapidly any modification of the specification. In this example modifications include classical algorithmic transforms such as loop unrolling and so on, but also structural transforms and organization of the code, which can have considerable effects on the performances. This shows that performing fast algorithmic characterization as proposed in our method is justified. Fig. 10 shows the elements of the processing setup. The encoder is based on a genetic algorithm and implemented on a server, we have not focused on this part. The decoding part can be implemented on several systems, including embedded systems. Fig. 10 shows the 4 main blocks of the decoding part.

Fig. 11 shows the results for the 4 main functions of the decoder. Clearly, "DecodeVideo" is the only function which includes some tests, limited however to 3%. We also notice that global memory accesses are frequent, this is due to the reads of the data from the video stream. "DecodeVideo" and "SetPixelValue" have the highest γ values, therefore they are to be examined first for optimization. These metrics have been computed with the first initial specification, these results have been used to refine the specification of the MP application [28], especially to increase the intrinsic parallelism of the "ComputeNorm" function.

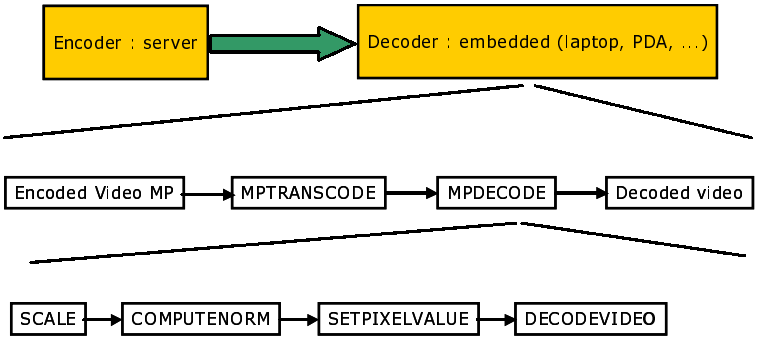

Figure 10: *Matching Pursuit setup (courtesy S. Bilavarn)*

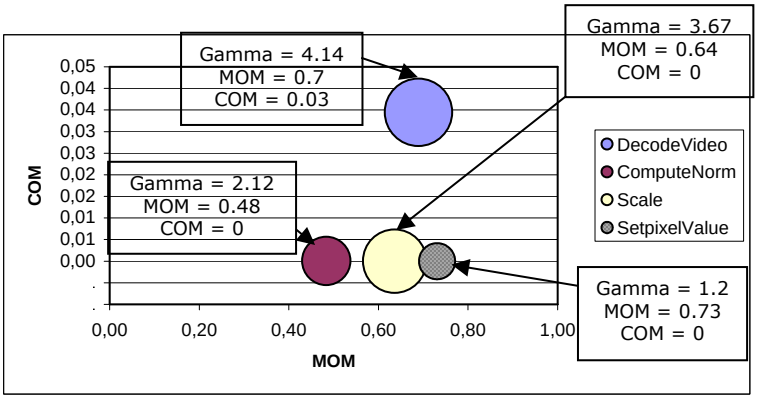

Figure 11: *Matching Pursuit characterization. γ is proportional to the size of the circle.*

5.6 Object Motion Detection (OMD)

This motion detection application have been developed by the LIST laboratory of the CEA research center [29] for the EPICURE project [21]. The typical target architecture is presented in Fig. 12. This is an intelligent video camera made of a CMOS sensor and a processor along with some reconfigurable logic. This application is typically embedded in video cameras and used for parking lot monitoring (detection of car and person moves), person counting in places such as subways and so on. We have used a large set of representative input data (from a parking lot monitoring appliance) to produce a profiling of the application functions used to fill the probability attributes of the graphs (cf.3.3). Fig. 13 illustrates how the OMD application works.

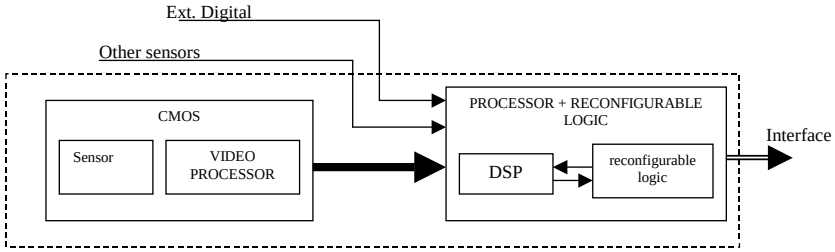

Figure 12: *Motion detection architecture. (Copyrights CEA).*

Figure 13: *OMD motion detection example. Video / background detection / moving object detection*

The OMD application is made of a hundred of functions. The main processing part is made of 31 functions, representing 1740 lines of C code. We have characterized these functions and found out that 16 of them are the most critical ones (i.e., those with the highest criticity metric (γ) values). These functions are those which should be optimized. The functions are quite complex, for example the function "Ic_gravityTest" is composed of 378 C code lines, translated into 2408 lines of HCDFG, the corresponding graph is made of 200 sub-graphs. The function "Ic_labelling" is made of about 200 lines of C code and 1200 lines in the HCDFG description. The results of the OMD characterization are presented in Fig. 14 and table 7. The possibility of characterizing an application rapidly is a important and very useful feature which enables the designer to sketch a

new architecture or to tune an existing one. In this case, all OMD functions have been characterized in a few seconds.

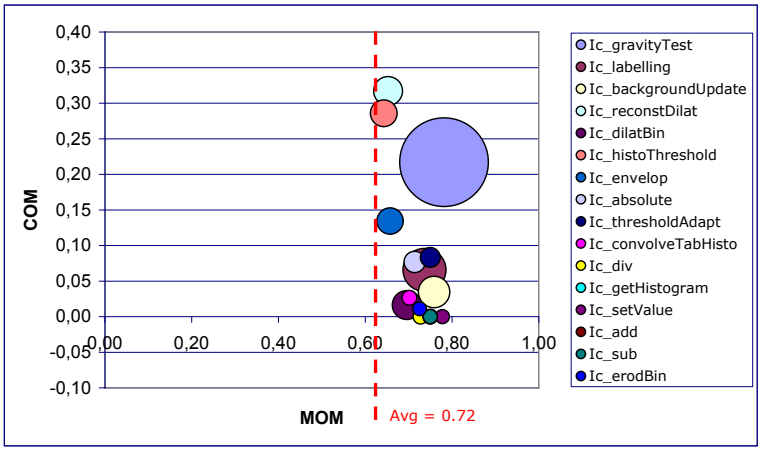

Figure 14: *OMD characterization. γ is proportional to the size of the circle.*

Solution number	**Function name**	**Critical Path (Nb cycles)**	**Gamma**	**MOM [0,1]**	**COM [0,1]**
1	Ic_gravityTest	2102	43,88	0,78	0,22
2	Ic_labelling	395269	10,31	0,74	0,07
3	Ic_BackgroungUpdate	73	5,62	0,76	0,03
4	Ic_reconstDilat	848144	4,75	0,65	0,32
5	Ic_dilatBin	49	4,69	0,70	0,02
6	Ic_histoThreshold	3	4,00	0,64	0,29
7	Ic_envelop	6098184	3,91	0,66	0,13
8	Ic_absolute	327683	2,60	0,71	0,08
9	Ic_thresholdAdapt	327683	2,20	0,75	0,08
10	Ic_convolveTabHisto	15879	1,27	0,70	0,03
11	Ic_div	524291	1,25	0,73	0,00
12	Ic_getHistogram	591622	1,22	0,75	0,00
13	Ic_setValue	1795	1,14	0,78	0,00
14	Ic_add	294919	1,11	0,75	0,00
15	Ic_sub	294919	1,11	0,75	0,00
16	Ic_erodBin	4776219	1,10	0,73	0,01

Table 7: *OMD characterization.*

The first observation which can be made is that all the functions have high MOM values, (0,72 on the average, which indicates that more than 2 operations out of 3 are memory accesses). This is due to the fact that there are numerous reads of data from the video stream and that the application is highly hierarchical (nested control structures for example), it also indicates that the inner DFGs are rather small so few local temporary data can be reused. This implies that the OMD application requires either a large local memory to store reused image data or high end input/output mechanisms (parallel data reading/writing).

Next we observe that γ variations are very significant since it evolves from 1,27 for "Ic_convolveTabHisto" up to 43.8 for "Ic_gravityTest". Using these values it is possible to sort the functions and to find out in what order they should be considered regarding the design of a specific architecture. Focusing on the most critical ones first enables to sketch an appropriate architecture and also to take reusing into account (the functions less critical can be implemented on existing resources allocated to the most critical ones). Finally COM values are comprised between 0 and 0,3 which denotes that that tests are not dominant (most of the control in the application is deterministic).

In the Epicure project we have performed the next steps after characterization, i.e., the scheduling/combinations/exploration step (Fig. 1) and a HW architectural estimation (HW projection in Fig. 1). In the overall design flow the characterization has been used to i) select the 16 most critical functions, ii) explore and detect the functions which have been implemented on FPGA (Xilinx V400EPQ2) and iii) choose a processor for the other functions (ARM922). The functions chosen for hardware implementation (Ic_gravityTest, Ic_labelling, Ic_BackgroundUpdate, Ic_dilatBin, Ic_envelop and Ic_absolute) are those which present high parallelism opportunities (high γ), high MOM and low MOC. In order to illustrate the usefulness of the metrics we present some results of the next steps of Design Trotter, namely the system-level estimation presented in table 8 computed with generic UAR and its projection on a Xilinx V400EPQ2 architecture given in table 9. The functions selected for hardware implementation by means of the metrics have been scheduled/combined/explored with Design Trotter. The results found during that step corroborate the indication given by γ and MOC: in terms of speed-up is has be found for example that Ic_gravityTest can be accelerated with factors up to 2614 as shown in table 8. Finally table 9 gives the results for the hardware projection of Ic_gravityTest on the Xilinx V400EPQ2 FPGA. The choice of the V400EPQ2 is based on the analysis of the metrics and the result of the scheduling/combination/exploration step, since its features suit well the need highlighted by the metrics and the system-level estimation.

6 Conclusion

In this paper we have proposed a high-level methodology and presented an interactive CAD tool which aims at guiding designers of embedded systems. More specifically, the framework enables the rapid characterization and exploration of applications specified using a usual standard language. The outcome is a set of metrics characterizing the application at all levels of hierarchy in terms of processing, control and memory orientation as well as in terms of potential parallelism. This information can be used in three ways:

Solution number	Nb cycles	Speed-up	ALU	MULT	Nb memory accesses	local memory size
1	10940	2614,85	2775	1	14020	25236
2	12634	2264,24	2755	1	13764	24788
3	13265	2156,53	2755	1	13764	24788
4	13461	2125,13	2755	1	13763	24788
5	22267	1284,70	2751	1	13743	24752
6	37979	753,22	791	1	3943	7112
7	41181	694,65	789	1	3933	7094
8	43925	651,26	789	1	3933	7094
9	49413	578,92	789	1	3933	7094
10	52157	548,47	789	1	3931	7094
11	54901	521,05	789	1	3929	7094
12	57645	496,25	789	1	3929	7094
13	60389	473,70	789	1	3928	7094
14	63133	453,11	789	1	3927	7094
15	71365	400,85	789	1	3927	7094
16	75828	377,25	397	1	1967	3566
17	85085	336,21	397	1	1967	3566
18	137221	208,47	397	1	1967	3566
19	139965	204,38	397	1	1966	3566
20	142709	200,45	396	1	1965	3566
21	145467	196,65	396	1	1964	3566
22	145582	196,50	395	1	1963	3564
23	264758	108,05	115	1	563	1044
24	529260	54,05	59	1	283	540
25	1055520	27,10	31	1	143	288
26	2414976	11,85	11	1	43	108
27	4302848	6,65	7	1	23	72
28	8009992	3,57	5	1	13	54
29	8547816	3,35	5	1	13	54
30	9623464	2,97	5	1	13	54
31	10161288	2,82	5	1	11	54
32	10699112	2,67	5	1	9	54
33	11236936	2,55	5	1	9	54
34	11774760	2,43	5	1	8	54
35	12312584	2,32	5	1	7	54
36	13926056	2,05	5	1	7	54
37	16615176	1,72	5	1	7	54
38	26833832	1,07	5	1	7	54
39	27371656	1,05	5	1	6	54
40	27909480	1,02	4	1	5	54
41	28447500	1,01	4	1	4	54
42	28447503	1,01	3	1	3	52
43	28592958	1,00	2	1	3	34
44	28592960	1,00	2	1	2	34
45	28606419	1,00	1	1	2	18
46	28606421		1	1	1	18

Table 8: *System-level scheduling/exploration of IC_gravityTest*

Solution number	Time (ns)	Nb states	Nb Logic Cells	Nb Dedicated Cells
25	20191042,08	4384	868	357
26	46196075,90	4415	428	191
27	82309179,39	857	340	56
28	153223136,97	614	296	36
31	194375278,15	618	296	32
36	266391525,22	625	296	28

Table 9: *IC_gravityTest hardware projection on a Xilinx V400EPQ2 FPGA*

i) when using a fixed architecture the specification characterization guides his algorithmic choices (e.g., parallel vs. sequential execution, loop unrolling, dedicated coprocessors, etc.)

ii) for a fixed specification the characterization guides his implementation choices (e.g., DSP vs. GPP vs FPGA).

iii) when neither the specification nor the architecture are definitely set, the designer can refine conjointly both aspects.

Therefore, by using our methodology the designer is guided, very early in the design process, for evaluating rapidly the impact of his algorithmic choices and choosing or building the most appropriate architecture for his application. This step is part of a high-level co-design environment called Design Trotter of which the goal is to assist designers of embedded systems such as SOCs.

We have presented two key points of this work: the first one is the HCDFG internal representation. This model is fully based on graphs and has been designed to fulfill our own requirements for the characterization and exploration of applications originally specified with the 'C' language. The second one is the characterization step itself. Two types of information are provided for each granularity level of the application functions. Firstly two orientation metrics are given, the MOM metric indicates how influent are data-transfers compared to data-processings, this point is usually related to the data-flow graph depth. The COM metric exhibits the weight of undesirable tests within the application. The MOM metric can be interpreted as a balance between the bandwidth and processing parallelism requirements, the MOC metric predicts the efficiency of spatial and temporal parallelisms. The second kind of metric produces the average level of parallelism comparatively to the critical path. Besides the information given about the available parallelism, this metric also shows up how it is ditributed over the different levels of granularity. Thus it indicates where large gain can be obtained with spatial parallelism but also where pipeline architecture is required.

We have illustrated these concepts with experiments conducted using the Design Trotter framework, which implements the C to HCDFG parser, the computation of the metrics and user interface for results analysis. By referring to the characterization results, designers of embedded systems can get rapidly feedback on their algorithmic choices and be guided in their architectural choices during the selection or the building of an appropriate architecture for the application.

The Design Trotter tool set has been designed as an open and flexible framework for implementing and testing new methods in the area of hardware / sofware codesign of embedded system, thus it constitutes an open space for future work.

References

[1] Y. Le Moullec, J-Ph. Diguet, and J-L. Philippe, "Design-trotter: a multimedia embedded systems design space exploration tool," in IEEE *Workshop on Multimedia Signal Processing (MMSP 2002)*, St. Thomas, US Virgin Islands, December 2002.

[2] Th. Gourdeaux, J-Ph. Diguet, and J-L. Philippe, "Design trotter: Interfunction cycle distribution step," in *11th Int. Conf. RTS Embedded Systems*, Paris, France, April 2003.

[3] S. Bilavarn, G. Gogniat, J.L. Philippe, and L. Bossuet, "Fast prototyping of reconfigurable architectures >from a c program," in *ISCAS 2003*, Bangkok, Thailand, May 2003.

[4] L. Bossuet, W. Burleson, G. Gogniat, V. Anand, A. Laffely, and J.L. Philippe, "Targeting tiled architectures in design exploration," in *10th Reconfigurable Architectures Workshop (RAW)*, Nice, France, April 2003.

[5] A. Azzedine, J-Ph. Diguet, and J-L. Philippe, "Large exploration for hw/sw partioning of multirate and aperiodic real-time systems," in *International Symposium on Hardware/Software Codesign (CODES)*, Estate Park, USA, May 2002.

[6] Gérard Berry, "The esterel primer," http://www-sop.inria.fr/meije/esterel/esterel-eng.html.

[7] N. Halbwachs, "Synchronous programming of rective systems," *Kluwer Academic Publisher*, 1993.

[8] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, *Hardware-Software Co-Design of Embedded Systems: The Polis Approach*, Kluwer Academic Publisher, June 1997.

[9] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of embedded systems: formal models, validation and synthesis," *Proceedings of the* I*EEE*, March 1997.

[10] J. Buck and E.E. Lee, ", the token flow model," in *Data Flow Workshop*, Hamilton Island, Australia, May 1992.

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

[12] "Systemc web page," http://www.systemc.org/.

[13] A. Gerstlauer and D. Gajski, "System level abstraction semantics," in *IEEE International Symposium on System Synthesis (ISSS)*, Kyoto, Japan, 2002.

[14] L.Cai, S.Verma, and D.D.Gajski, "Comparison of specc and systemc languages for system design," Tech. Rep. CECS-03-11, Univ. of California, Irvine, Irvine, USA, 2003.

[15] "Esterel studio web page," http://www.esterel-technologies.com/.

[16] L. Guerra, M. Potkonjak, and J. Rabaey, "System-level design guidance using algorithm properties," in IEEE *Workshop on* VLSI *Signal Processing*, San Diego, USA, October 1994.

[17] J-Ph. Diguet, O. Sentieys, J-L. Philippe, and E. Martin, "Probabilistic resource estimation for pipeline architecture," in IEEE *Workshop on* VLSI *Signal Processing*, Sakai, Japan, October 1995.

[18] F. Vahid and D. D. Gajski, "Closeness metrics for system-level functional partitioning," in *EDAC*, Brighton, U.K., September 1995, pp. 328–333.

[19] L. Carro, M. Kreutz, F. Wagner, and M. Oyamada, "System synthesis for multiprocessor embedded applications," in *Design Automation and Test in Europe Conference (DATE)*, Paris, France, March 2000.

[20] D.Sciuto, F.Salice, L.Pomante, and W.Fornaciari, "Metrics for design space exploration of heterogeneous multiprocessor embedded systems," in *International Symposium on Hardware/Software Codesign (CODES)*, Estes Park, USA, May 2002.

[21] M.Auguin, K.Ben Chehida, J-Ph.Diguet, X.Fornari, A-M.Fouilliart, C.Gamrat, G.Gogniat, P.Kajfasz, and Y.Le Moullec, "Partitioning and CoDesign tools & methodology for Reconfigurable Computing: the EPICURE philosophy," in 3^{rd} *Int. Work. on Systems, Architectures, Modeling Simulation (SAMOS03)*, Greece, July 2003.

[22] A.Dasdan, D.Ramanathan, and R.K.Gupta, "Rate derivation and its application to reactive real-time embedded systems," in 35^{th} ACM/IEEE *Design Automation Conf.*, San Francisco, USA, 1998.

[23] P.Grun, F.Balasa, and N.Dutt, "Memory size estimation for multimedia applications," in 6^{th} *Int. Work. on HW/SW Co-Design*, Seattle,USA, Mar. 1998.

[24] M. Miranda, M. Janssen, F. Catthoor, and H. De Man, "ADOPT: Efficient Hardware Address Generation in Distributed Memory Architectures," in 9^{th} IEEE/ACM *Int. Symp. on System Synthesis*, La Jolla, USA, November 1996.

[25] S. Wuytack, J-Ph. Diguet, F. Catthoor, and H. De man, "Formalized methodology for data reuse exploration for low-power hierarchical memory mappings," IEEE *Transaction on* VLSI *Systems*, vol. 6, no. 4, pp. 529–537, December 1998.

[26] Y. Le Moullec, J-Ph. Diguet, D. Heller, and J-L. Philippe, "Fast and adaptive data-flow and data-transfer scheduling for large design space exploration," in *Great Lakes Symposium on* VLSI*(GLSVLSI)*, New-York, USA, April 2002.

[27] M.Antonini, M.Barlaud, P.Mathieu, and I.Daubechies, "Image coding using wavelet transform," *IEEE Transaction on Image Processing*, vol. 1, no. 2, pp. 205–206, April 1992.

[28] S.Bilavarn, P.Vandergheynst, and E.Debes, "Methodology and tools to define special purpose processor architecture," http://ltswww.epfl.ch/ bilavarn, 2003, Intel grant 11409.

[29] L. Letellier and E. Duchesne, "Motion estimation algorithms," Tech. Rep., L.C.E.I, C.E.A, Saclay, France, 2001.