Journal of VLSI Signal Processing 42, 185–208, 2006

DOI: 10.1007/s11265-005-4181-x

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

YANNICK LE MOULLEC, JEAN-PHILIPPE DIGUET, NADER BEN AMOR, THIERRY GOURDEAUX AND JEAN-LUC PHILIPPE
LESTER, University of South Brittany, France, F-56321 Lorient Cedex

Received October 23, 2003; Revised January 3, 2005; Accepted February 23, 2005

Abstract. Designing embedded systems is a challenging task during which wrong choices can lead to extremely costly re-design loops, especially when these wrong choices are made during the algorithm specification and the mapping over the selected architecture. In this paper we propose a high-level approach for design space exploration, using a usual standard language as input. More precisely we present the two first steps of the Design Trotter framework: (i) the specification step and its underlying internal model (HCDFG: Hierarchical and Control Data Flow Graph) and (ii) the characterization step which takes place very early in the design flow. Indeed, once transformed into our internal representation, the specification is rapidly and automatically characterized and explored at the algorithmic level. The framework provides the designer with metrics so that he can evaluate, very early in the design process, the impact of algorithmic choices on resource requirements in terms of processing, control, memory bandwidth and potential parallelism at different levels of granularity. The overall aim of our approach 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 the Design Trotter framework in order to select or build suitable architectures for specific applications.

Key words: specification, characterization, algorithm/architecture matching, design-space exploration, SoCs

1. Introduction

The context of our work is the hardware/software co-design of embedded multimedia applications. In this domain, 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 on the two aspects, using feedbacks between them. In all cases the designer needs relevant knowledge about the specification in terms of 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 from the tedious task consisting in the evaluation of 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 (currently the C language) for specifying the application. This specification is then automatically transformed into a completely graph-based model (HCDFG) 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 5) 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 perform automatically and rapidly the tedious algorithmic exploration for the functions called from the event-based level.

1.1. Overview of the Design Trotter Framework

The work presented in this paper is part of a complete design framework called Design Trotter [1]. Design Trotter is a set of cooperative tools which aim at guiding embedded system designers early in the design 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 explored separately. For each function, the two first steps (presented in this paper) include the construction of the graph (HCDFG model) and the characterization by means of metric computations. Then a scheduling step, presented in [1] is performed for all the data-flow graphs and loop nests within the functions. The design space exploration is performed by means of a large set of time constraints (e.g., from the critical path to the sequential execution). Finally, the results are combined to produce trade-off curves (number of resources vs. number of 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 some of the 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].

The rest of the article is organized as follows: in Section 2 we give an overview of existing specification models and characterization tools. We expose the contributions of our paper for both specification, internal representation model and characterization aspects. Sections 3, 4 and 5 detail these points respectively. In Section 6 we present some results of the characterization step showing the interest of the proposed method. Finally in Section 7 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 5.

2.1.1. Existing Work for Specification. Generally, existing design approaches consider only a partial design flow. The decomposition of the design flow results from the architectural target and/or from the type of data processed by the application. We can mention control oriented models [6–9], data-flow models [9, 10], Khan process networks [11], and Ptolemy domains (e.g., DE, DF) [12].

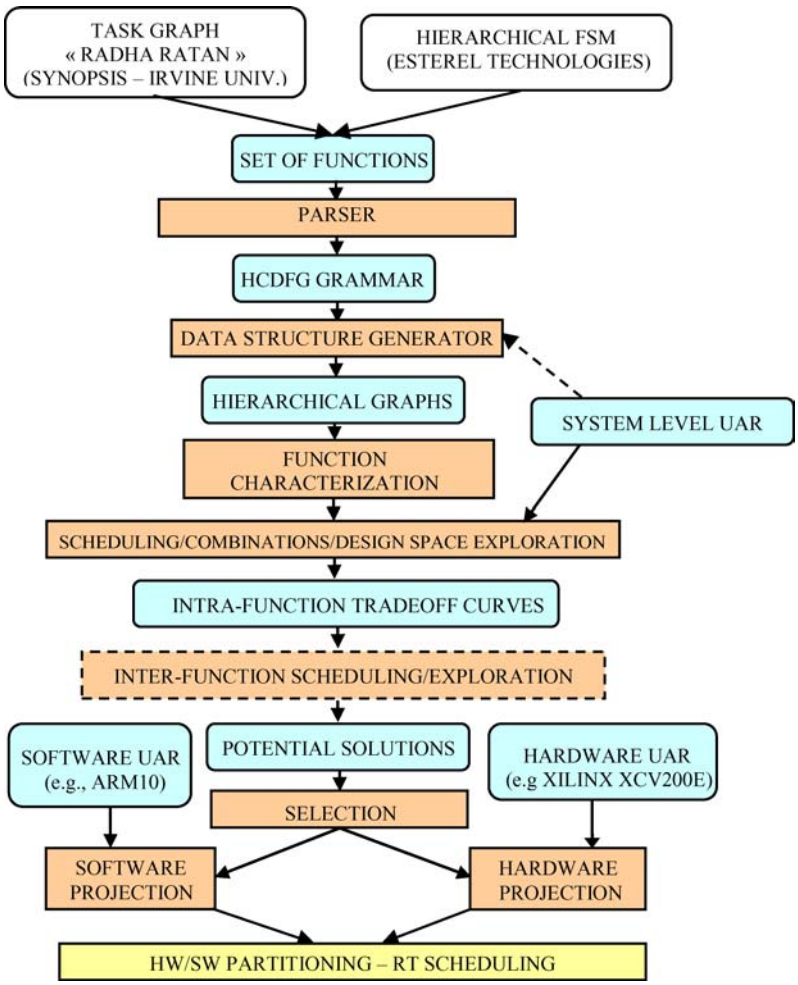

Figure 1. Design Trotter design flow.

To alleviate this problem, some attempts to define a unified modeling and design framework have been proposed. They are generally based on co-simulation approaches (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 hardware synthesis and assembly code for processors.

SystemC [13] is a design and verification language enabling the description of systems from high-levels of abstraction (e.g., algorithmic-level) down to hardware and software implementation levels. The modeling features of SystemC are provided as a C++ class library. SystemC is a good 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 [14], 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 [15] aim at providing designers with a common language for progressively modeling an application by means of refinement, from the un-timed system-level to the cycle-accurate implementation 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 between the two models is not necessarily clear and can vary depending on the designer background.

The main contribution of this work in terms of specification is the internal representation model "HCDFG". The requirements of the tools included in the Design Trotter framework have guided our choice towards a suitable internal representation model. This model should have the following features:

- Hierarchy: enables a 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. Two types of attributes are considered: those created during the parsing step (e.g., data-type) and those created during the characterization and exploration steps (e.g., metrics);
- Parallelism specification: parallelism is a key feature for 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 [16] 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 the Esterel framework) and tools for formal proof, 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 the results of metric computations (cf. 5), 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-based representation of the specification. 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 Section 4).

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 and present our contribution.

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 need for 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 and 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 and multi-granularity: enable the characterization of the different granularity levels: e.g., (i) loop body, (ii) loop, (iii) sequence of loops; they also offer 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. Works dealing with metrics in the domain of high-level synthesis [17, 18] and hardware software co-design [19–21] have been recently proposed. In [17] 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 [18], some probability-based metrics are proposed to quantify the communications between arithmetic operators (through memory or registers). These metrics focus on a fine grain analysis and are mainly used to guide the design of datapaths, especially to optimize local connections and resource reuse. The metrics from [19] 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 [20]. 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 that framework a coarse and fixed granularity level is considered and the target is limited to predefined processors. Moreover the technique does not take instruction dependencies into account 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 [21 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, etc.), 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 [20] these metrics are dedicated to HW/SW partitioning, they do not exploit instruction dependencies and address a fixed 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 these requirements by means of new metrics which are detailed in Section 5.

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 4.4) that can be more or less specialized to either offer 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 for each level of granularity and thus are naturally available at all levels, from the leaf DFGs (Data-Flow Graph) to the complete HCDFG. Finally, the system metrics from [21, 20] 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 5.

3. Specification

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. In both cases the aim of the HCDFG model is to represent the lower-level functions called by the HFSM or TG models, such that their estimation and exploration are facilitated. The two specification models are rapidly presented in Sections 3.1 and 3.2. The HCDFG model is detailed in Section 4.

3.1. HFSM-HCDFG

This type of specification is presented in Fig. 2. The HFSM model has been used in the EPICURE project [22]. The hierarchical decomposition at the system-level is made with Esterel studio [16]. 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 when he considers that the HFSM model is no longer suitable. 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 concurrency. 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., the 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 [23] to specify the task-graph.

4. The HCDFG Model

4.1. HCDFG Basics

As previously mentioned, 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. From a practical point of view, a HCDFG is automatically built for each C function called by the HFSM-HCDFG or TG-HCDFG models presented in the previous section. Note that it is also possible to start the design process by specifying directly C functions. 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 space exploration. Namely a HCDFG is built 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;
- The initial hierarchical structure is used in order to preserve data locality;
- The code hierarchy is also exploited to perform a component based approach by means of graph pattern detection. A given C function/block is seen as a single graph-component (HCDFG) that can then be instantiated several times;
- 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 characterization and estimation steps, important elements and structures can be easily identified by means of our uniform model.

4.2. 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. A HCDFG is composed of elementary nodes (processing, memory, control), dependence edges (control, data) and graphs that can be hierarchical. A HCDFG example is given in Fig. 4. The different key features are detailed hereafter.

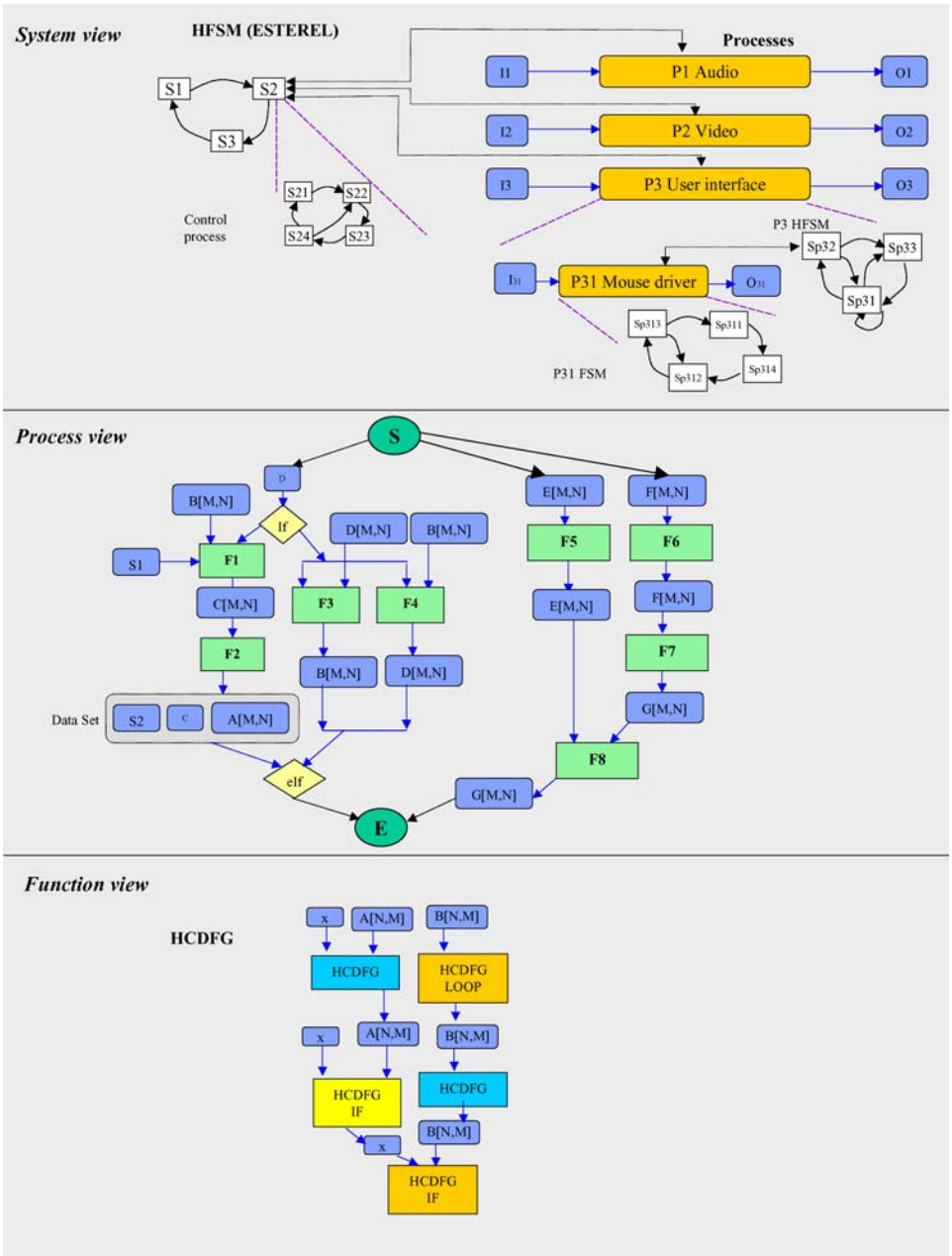

Figure 2. Specification example with the HFSM-HCDFG model.

4.2.1. 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, etc.). As the association between operations and resources is defined in the UAR file (described in Section 4.4), 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. 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

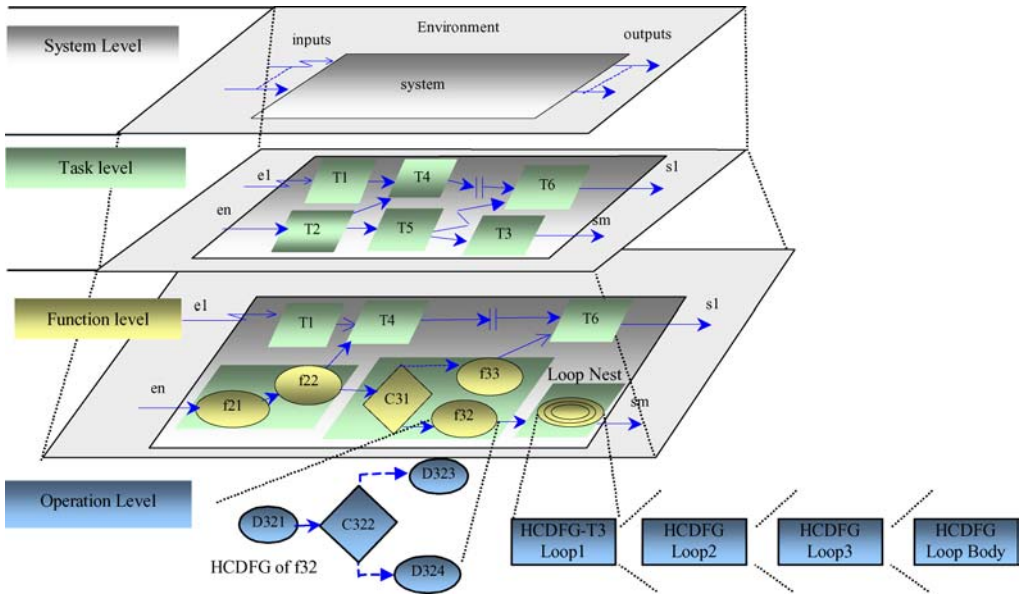

Figure 3. Specification example with the TG-HCDFG model.

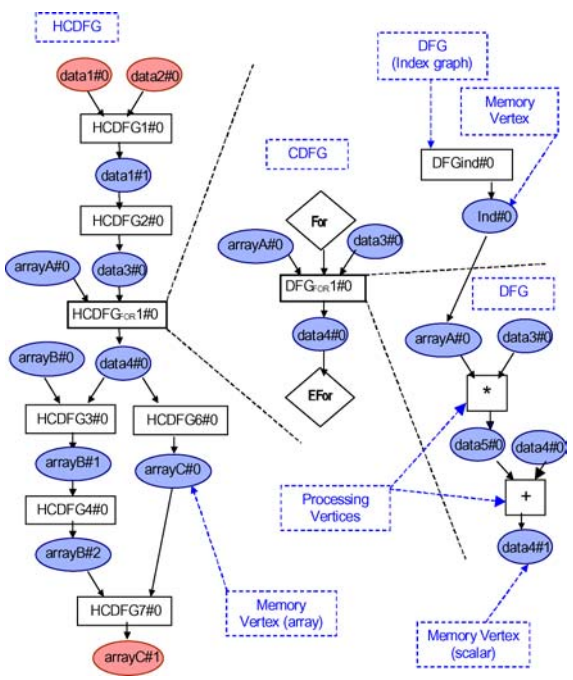

Figure 4. Elements of a HCDFG.

type (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 proposed model avoids the duplication of the information required to represent data and accesses. For multi-dimensional data (arrays, vectors), addressing mechanisms are explicitly represented in the graph through index DFGs. Memory vertices (scalars and arrays) are exposed on the right and left part of Fig. 4 respectively.

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

4.2.2. 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 that 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.

4.2.3. 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 FOR structures, and finally one or more H/CDFGs which represent the processing conditioned by the control node. Moreover, an attribute is used to store the execution probability of each branch. The probabilities can be obtained by means of 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 mechanisms are allowed at the moment). 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.

4.3. 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*.

4.4. 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 functionalities) can be described and associated to operations/functions of the HCDFG description. Regarding the memory part, the user defines the number of hierarchy levels (L_i) and the number of cycles (L_i) associated to each type of access.

Figure 5 shows an example of two UAR 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.

5. Characterization

The functions composing an application can have very different features in terms of orientation (processing, control, memory) and potential parallelism. The characterization of the 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:

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

```
<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).

5.1. Definitions and Memory Model

Before to detail the metrics, we introduce some terms which will be used in what follows.

- *Processing*: includes (i) computation operations (ALU, MAC, etc.), (ii) address computations which can be performed on ALUs or specific units such as AGUs (*Address Generator Unit*) [24], and iii) 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 take into account: 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 memory 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 the RAM and the ROM.

5.1.1. 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. Several types of memory nodes are taken into account (the type is stored as an attribute for each memory node):

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

Table 1. Abbreviation used.

Abbreviation	Meaning
Np	number of processing operations = number of operations of type ALU, Macs, etc. + Index computation + deterministic tests
Nc	number of non-deterministic tests (control operations which can not be eliminated at compile time)
Nm	number of global memory accesses operations
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 "FOR" and "IF" structures
evol	graph of the evolution part of a "FOR" structure
N_{ite}	number of iterations in a loop structure

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. 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].

5.2. *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 graphs (CDFGs) and hierarchical graphs (HCDFGs) using mutual-exclusive rules (for CDFGs) and parallel and sequential rules (for CDFGs and HCDFGs). The metrics at level i of the hierarchy are not simply obtained by combining the metrics of 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 5.3.1, 5.3.2, 5.4.3, and 5.4.3.

Note also that on the one hand the orientation metrics (Section 5.3) are computed without knowledge of the node ASAP/ALAP dates, only by counting the relevant elements. On the other hand the criticity (Section 5.4) and (H)DRM (Section 5.5) metric computations use ASAP/ALAP dates computed with UAR features.

5.3. *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 computations for 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.

5.3.1. 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 indicate 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, etc.) as well as an efficient use of memory hierarchy and data locality [25].

- *MOM computation for a basic block (DFG).* For a DFG, MOM is computed as follows:

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

- *MOM computation for a IF-THEN-ELSE structure (and SWITCH by extension).* For this structure, MOM is defined 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.

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

- *MOM computation for a FOR structure (and WHILE DO-WHILE by extension).* For this structure, MOM is defined 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.

$$\mathrm{MOM}_{\mathrm{FOR}} = \frac{Nm_{\mathrm{eval}} + Nm\mathrm{for} + Nm_{\mathrm{evol}}}{\sum_{x=p,c,m}(Nx_{\mathrm{eval}} + Nx_{\mathrm{for}} + Nx_{\mathrm{evol}})}$$

- *MOM computation for a HCDFG.* For a HCDFG, MOM is defined 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.

$$\mathrm{MOM}_{\mathrm{HCDFG}} = \frac{\sum_{\mathrm{sub-graphs}\,j} Nm_j}{\sum_{\mathrm{sub-graphs}\,j, x=p,c,m} Nx_j}$$

5.3.2. Control Orientation Metric (COM). COM indicates the 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 the global memory. COM values are bounded in the interval [0;1]. The more COM $\rightarrow$ 1, 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. For a CDFG, the generic formula for COM is as follows:

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

- *COM computation for a basic block (DFG).* For a DFG, COM equals 0 since there is no control in a DFG.
- *COM computation for a IF-THEN-ELSE structure (and SWITCH by extension).* For this structure, COM is defined 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.

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

- *COM computation for a FOR structure (and WHILE DO-WHILE by extension).* For this structure, COM is defined 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.

$$\mathrm{COM}_{\mathrm{FOR}} = \frac{Nc_{\mathrm{eval}} + Nc\mathrm{for} + Nc_{\mathrm{evol}}}{\sum_{x=p,c,m}(Nx_{\mathrm{eval}} + Nx_{\mathrm{for}} + Nx_{\mathrm{evol}})}$$

- *COM computation for a HCDFG.* 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.

$$\mathrm{COM}_{\mathrm{HCDFG}} = \frac{\sum_{\mathrm{sub-graphs}\,j} Nc_j}{\sum_{\mathrm{sub-graphs}\,j, x=p,c,m} Nx_j}$$

5.4. 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 metric "γ" such as:

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

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 (i.e., for a H/CDFG) is computed hierarchically by combining the critical path of its sub-parts.

γ indicates the average parallelism available at a specific hierarchy level: let's 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 the value of the COM metric. 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.

5.4.1. γ Computation for a IF-THEN-ELSE Structure (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.

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

5.4.2. γ Computation for a FOR Structure (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.

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

5.4.3. γ Computation for a HCDFG: For a HCDFG representing a serial execution of sub-graphs, the criticity metric is defined 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 parallel 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.

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

5.5. DRM and HDRM Metrics

5.5.1. DRM Metric. Balancing processing and data-transfer operations is a critical point in system design to face the memory bandwidth bottleneck (as compared to the processor performances). The balance can be obtained by means of scheduling algorithms adapted to the function orientation. In order to guide the analysis of the functions and to select the most appropriate DFG scheduling scheme, we have defined a metric called *DRM: Data Reuse Metric*. At system-level the only memory hierarchy information available is 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 automatically. The estimation is performed as explained in the next paragraph. The DRM metric indicates the ratio between the global and local memory accesses. A local access which produces a memory conflict (local memory full) involves read and write accesses to the global memory, thus the number of extra global accesses is $\beta \times$ EGT where β is the average number of cycles required to access the global memory. For example, a single main memory which requires only one cycle per access implies that $\beta = 2$ (read plus write).

We use the average data lifetime to estimate the quantity of data alive per 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, 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:

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

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

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

where T 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:

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

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

Let's consider a memory hierarchy with the following characteristics:

Level 1 (L1): $latency1(l1) = 1cycle$, L2: $l2 = 2cycles$, L3: $l3 = 3cycles$. Let's assume that 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 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 defined as:

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

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

5.5.2. 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. 6: A1 and A2 are the amount of exclusive input data read by HCDFG1 and HCDFG2 respectively. A12 quantifies the input data common to both graphs and A3 is the amount of result data from the first graph transmitted to the second one. Thus the HDRM provides the ratio of reused data between two HCDFG which can be parallel (A3 = 0), sequential (A12 = 0) or a combination.

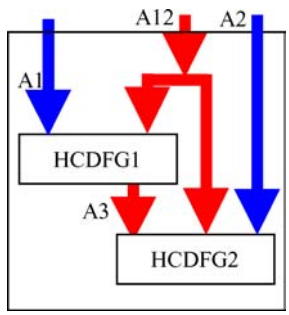

Figure 6. Illustration of the HDRM metric.

The HDRM metric is computed as follows:

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

HDRM = 1 means that the reuse ratio is maximal: all data read by 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 reuse has an important impact because the optimization opportunities are mainly due to memory management of loop nests: the HRDM metric can be computed by considering HCDFG1 and HCDFG2 as two successive loop iterations.

5.5.3. HDRM 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 Fig. 6, 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

Table 2. HDRM metric Motion estimation theoretical results.

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	$(2m + n - 1)^2 n$
Frame row	1408	1408	1408	1408	5456	25336	0	0.87	$W * (2m + n - 1)$

OF: Old Frame, NF: New Frame

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 W×H 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 stand for 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.

5.6. *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.

6. Experimental Results

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

6.1. *DWT*

The DWT algorithm [27] has been implemented using the lifting scheme. The C code is made of one function (translated into a HCDFG, level N) englobing the code for the sub-blocks. 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 lower level of granularity depend on the structure of each sub-blocks whose metrics are automatically computed exactly like for the upper levels. The time required for the whole characterization step is 500 ms on an Intel PIII@700Mhz. 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: 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 70% of the operations are data accesses. So the application is clearly, at all levels, memory oriented. Finally, γ values are approximately 1.5 for all the functional blocks. The horizontal and vertical blocks have γ 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 the spatial parallelism is rather weak considering the fine

Table 3. DWT characterization.

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	0.724	0	2.704
Vlines	0.724	0	2.704
Top-graph *N*	MOM *N*	COM *N*	γ *N*
DWT (top graph)	0.765	0	2.704

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 (i) there is no need for complex control structures, (ii) there are important needs for high data-access requirements and (iii) there is a coarse grain parallelism available. 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.

6.2. 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 test. 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

Table 4. G722 characterization.

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
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

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.

6.3. 2D DCT

This application is a well known 2D-DCT for 8×8 image blocks. 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 can be observed in Table 5. We can notice that the γ metric reflects 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 approximately 0.5. It means that the reuse of temporary local data is here more important than for the DWT example; it is also related to the larger degree of available parallelism.

6.4. 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 results 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 33%) also indicate an important data accesses frequency. It means that these functions are control-oriented and also 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 because the functional blocks are clearly control oriented. Note that another very efficient architecture could be implemented using a dedicated FSM associated with fast FIFOs.

Table 5. 2D DCT characterization.

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*
DCT8×8 (top graph)	0.575	0	5.714

Table 6. TCP characterization.

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	375	0.06
TCPSENT	0.508	0.320
TCPRESET	0.667	0.148

6.5. Matching Pursuit

We have studies the Matching Pursuit (MP) application in the context of a collaboration project with the EPFL [28]. The matching pursuit application is a new compression scheme 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. Figure 7 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. Figure 7 shows the 4 main blocks of the decoding part.

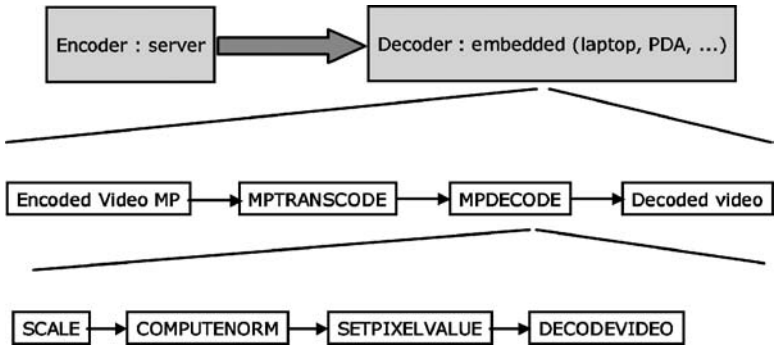

Figure 7. Matching pursuit setup (courtesy S. Bilavarn).

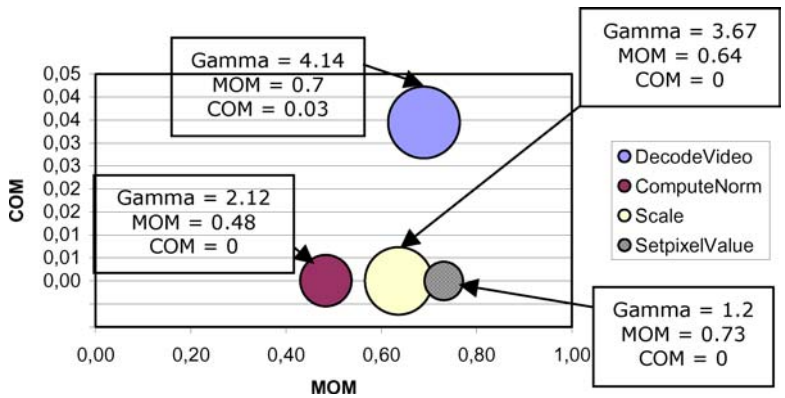

Figure 8. Matching Pursuit characterization. γ is proportional to the size of the circle.

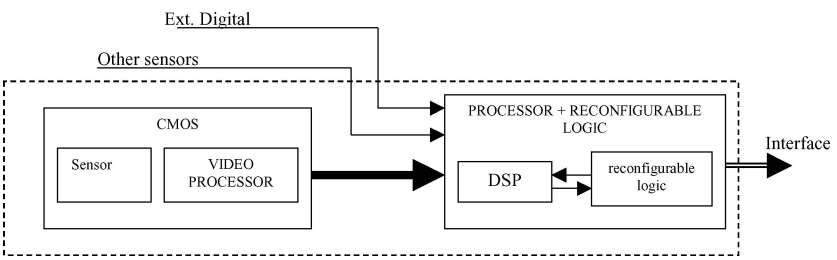

Figure 9. Motion detection architecture. (Copyrights CEA).

Figure 10. OMD motion detection example. Video/background detection/moving object detection.

Figure 8 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 have to be examined first for optimization. Once these metrics have been computed with the first initial specification, the results have been used to refine the specification of the MP application [28], especially to increase the intrinsic parallelism of the "ComputeNorm" function.

6.6. *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 A.Dasdan, D.Ramanathan, and [22]. The typical target architecture is presented in

Table 7. OMD characterization.

Function 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
12	Ic_getHistogram	591622	1.22	0.75	0
13	Ic_setValue	1795	1.14	0.78	0
14	Ic_add	294919	1.11	0.75	0
15	Ic_sub	294919	1.11	0.75	0
16	Ic_erodBin	4776219	1.10	0.73	0.01

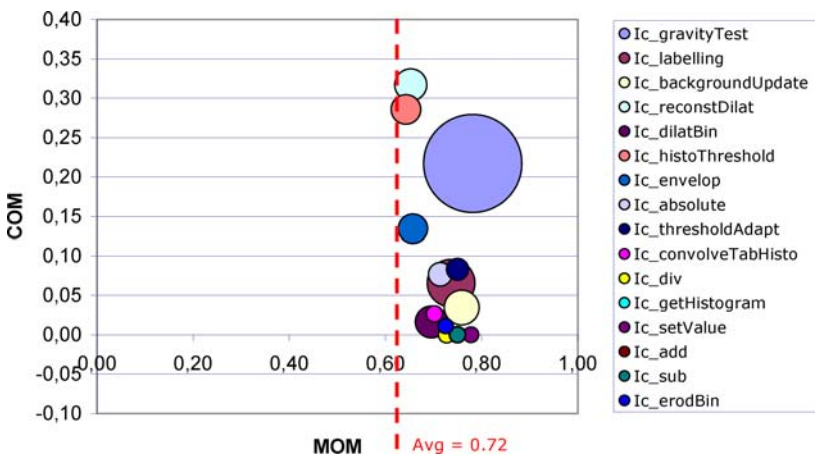

Figure 11. OMD characterization. γ is proportional to the size of the circle.

Table 8. System-level scheduling/combinations/exploration of IC_gravityTest.

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	7172
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	2	1	3	34

Table 8. Continued.

Solution number	Nb cycles	Speed-up	ALU	MULT	Nb memory accesses	Local memory size
44	28592960	1	2	1	2	34
45	28606419	1	1	1	2	18
46	28606421	1	1	1	1	18

Fig. 9. This is an intelligent video camera composed 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 application) to produce a profiling of the application functions used to fill in the probability attributes of the graphs. Figure 10 illustrates how the OMD application works.

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. 11 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. Each OMD function has been characterized within a few seconds on an Intel PIII@700 MHz.

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 little 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.

Next, we observe that the values of γ are significantly different for the functions composing the OMD application: these values range from 1.27 for "Ic_convolveTabHisto" up to 43.8 for "Ic_gravityTest". By referring to these values the designer can sort the functions and find out in which 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 hardware reuse into account (the less critical functions 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 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 (see Fig. 1) and a HW architectural estimation (called 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 the 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

Table 9. IC_gravityTest hardware projection on a Xilinx V400EPQ2 FPGA.

Solution number(ns)	Time	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

Ic_absolute) are those which present high parallelism opportunities (high γ), high MOM and low MOC. The results of the hardware projection step corroborate the indication given by γ and MOC: in terms of speed-up is has be found that Ic_gravityTest can be accelerated with a factor 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.

7. Conclusion

In this paper we have proposed a high-level methodology which aims at guiding designers of embedded systems. More specifically, the framework enables the rapid characterization and exploration of applications specified using a 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:

(i) when using a fixed architecture, the specification characterization guides the algorithmic choices (e.g., parallel vs. sequential execution, loop unrolling, dedicated co-processors, etc.)
(ii) for a fixed specification the characterization guides the 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 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. We have presented two key points of this work: the first one is the HCDFG internal representation. This model is based on graphs and has been designed to fulfill our own requirements for the characterization and exploration of embedded applications. The second point is the characterization step itself. Two types of information are provided for each granularity level of the application functions. Firstly two orientation metrics are provided: the MOM metric indicates how influent are data-transfers compared to data-processing, 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 the balance between the bandwidth and processing parallelism requirements whereas the MOC metric predicts the efficiency of spatial and temporal parallelisms. The second kind of metric (γ *gamma*) indicates the average level of parallelism comparatively to the critical path. Besides the information given about the available parallelism, this metric also indicates how it is distributed over the different levels of granularity. Thus it indicates where large gain can be obtained with spatial parallelism but also where a pipelined 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 graphical user interface for analyzing the results. By referring to the characterization results, designers of embedded systems can rapidly get feedback on their algorithmic choices and can 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 co-design 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," in *Proceedings of the IEEE*, 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, 2001, pp. 57–63.
12. "Ptolemy project webpage," http://ptolemy.eecs.berkeley.edu/.
13. "Systemc web page," http://www.systemc.org/.
14. A. Gerstlauer and D. Gajski, "System level abstraction semantics," in *IEEE International Symposium on System Synthesis (ISSS)*, Kyoto, Japan, 2002.
15. 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.
16. "Esterel studio web page," http://www.esterel-technologies.com/.
17. L. Guerra, M. Potkonjak, and J. Rabaey, "System-level design guidance using algorithm properties," in *IEEE Workshop on VLSI Signal Processing*, San Diego, USA, Oct. 1994.
18. 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, Oct. 1995.
19. F. Vahid and D. D. Gajski, "Closeness metrics for system-level functional partitioning," in *EDAC*, Brighton, U.K., Sept. 1995, pp. 328–333.
20. 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.
21. 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.
22. 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 3rd Int. Work. on Systems, Architectures, Modeling Simulation (SAMOS03), Greece, July 2003.
23. A. Dasdan, D. Ramanathan, and R.K. Gupta, "Rate derivation and its application to reactive real-time embedded systems," in *35th Acm/IEEE Design Automation Conf.*, San Francisco, USA, 1998.
24. M. Miranda, M. Janssen, F. Catthoor, and H. De Man, "ADOPT: Efficient Hardware Address Generation in Distributed Memory Architectures," in *9th IEEE/Acm Int. Symp. on System Synthesis*, La Jolla, USA, Nov. 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, 1998, pp. 529–537.
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, 1992, pp. 205–206.
28. S. Bilavarn P. Vandergheynst E. Debes, "Methodology and Tools to Define Special Purpose Processor Architecture," http://ltswww.epfl.ch/ bilavarn/SpecialPurposeProcessor_files/frame.htm, 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.

Yannick Le Moullec received the M.S degree from the University of Rennes (France) in 1999 and the Ph.D degree from the University of South Brittany (France) in 2003. His thesis focused on Hardware/Software codesign, and more specifically on high-level design space exploration of embedded systems. His thesis work is part of the EDA framework "Design Trotter". In 2003 he joined the center for embedded software (CISS) at Aalborg University (Denmark) as a post-doc fellow. His current work deals with design space exploration for wireless applications.

Jean-Philippe Diguet received the M.S degree and the Ph.D degree from Rennes University (France) in 1993 and 1996 respectively. His thesis, carried out at LASTI lab., focused on the estimation of hardware complexity and algorithmic transforms for architectural synthesis. Then he joined the IMEC in Leuven (Belgium) where he worked as a post-doc fellow on the minimization of the power consumption of memories at the system-level. He has been a member of the LESTER lab. (Lorient, France) since 1998, where he initiated a

research project in design space exploration both at the algorithmic and system levels. He has been an associated professor at the University of South Brittany (France) from 1998 until 2002. In 2002, he initiated a technology transfer and co-funded Dixip, a company in the domain of wireless embedded systems. Since 2004 he is a CNRS (French National Scientific Research Center) researcher. His work focuses on two topics. The first one is the management of the development of the EDA framework "Design Trotter" for design space exploration and combining the design space exploration with technology mapping over reconfigurable architectures. The second topic is the definition of environment-aware, predictive and self-adaptive architectures under QoS and power constraints, this topic includes new OS services, NOCs and architecture reconfiguration control.
Jean-Philippe.Diguet@univ-ubs.fr

Nader Ben Amor is a PhD student at both University of South Brittany (France) and Sfax Engineering School (Tunisia). He received the Electrical Engineering Diploma in 2000 and the M.S Degree in Electronics in 2001 from Sfax Engineering School. His research interest is the design of embedded multimedia processors.

Thierry Gourdeaux received the Engineer Title and the M.S degree from the ENSSAT school, Lannion (France), in 1997. Since 1999 he is a research engineer at the University of South Brittany. His work includes the analysis of DSP systems for spatial applications (with Alcatel Espace), technology and knowledge transfers to the industry community and the development of EDA tools such as "Design Trotter" and "Soft Explorer".

Jean-Luc Philippe is currently a full Professor at the University of South Brittany (France). He is leading a research group working on design methodologies for electronic systems executing under performances, consumption and safety constraints. Once implemented into tools, these methodologies are applied to several domains: signal and image processing, telecommunications, transitic systems, etc.